December 2017

Competitive Power Down Methods in Green Computing

James Andro-Vasko
University of Nevada, Las Vegas, vaskodagamer@gmail.com

COMPETITIVE POWER DOWN METHODS IN GREEN COMPUTING

by

James Andro-Vasko

Master of Science (M.Sc.)
University of Nevada, Las Vegas
2011

Bachelor of Science (B.Sc.)
University of Nevada, Las Vegas
2009

A dissertation submitted in partial fulfillment of
the requirements for the

Doctor of Philosophy Degree - Computer Science

Department of Computer Science
Howard R. Hughes College of Engineering
The Graduate College

University of Nevada, Las Vegas
December 2017

UNLV | GRADUATE COLLEGE

Dissertation Approval

The Graduate College
The University of Nevada, Las Vegas

November 16, 2017

This dissertation prepared by

James Andro-Vasko

entitled

Competitive Power Down Methods in Green Computing

is approved in partial fulfillment of the requirements for the degree of

Doctor of Philosophy Degree - Computer Science
Department of Computer Science

Wolfgang Bein, Ph.D.
Examination Committee Chair

Kathryn Hausbeck Korgan, Ph.D.
Graduate College Interim Dean

Lawrence Larmore, Ph.D.
Examination Committee Member

Ajoy Datta, Ph.D.
Examination Committee Member

Andreas Stefik, Ph.D.
Examination Committee Member

Robert Boehm, Ph.D.
Graduate College Faculty Representative

Abstract

For the power-down problem one considers a device which has states OFF, ON, and a number of intermediate states. The state of the device can be switched at any time. In the OFF state the device consumes zero energy and in the ON state it works at its full power consumption. The intermediate states consume only some fraction of energy proportional to the usage time but switching back to the ON state has has different constant setup cost depending on the current state. Requests for service (i.e. for when the device has to be in the ON state) are not known in advance, thus power-down problems are studied in the framework of online algorithms, where a system has to react without knowledge of future requests. Online algorithms are analyzed in terms of competitiveness, a measure of performance that compares the solution obtained online with the optimal online solution for the same problem, where the lowest possible competitiveness is best.

Power-down mechanisms are widely used to save energy and were one of the first problems to be studied in green computing. They can be used to optimize energy usage in cloud computing, or for scheduling energy supply in the smart grid. However, many approaches are simplistic, and do not work well in practice nor do they have a good theoretical underpinning. In fact, it is surprising that only very few algorithmic techniques exist. This thesis widens the algorithmic base for such problems in a number of ways. We study systems with few states which are especially relevant in real wold applications. We give exact ratios for systems with three and five states. We then introduce a new technique, called decrease and reset, where the algorithm automatically attunes itself to the frequency of requests, and gives a better performance for real world inputs than currently existing

algorithms. We further refine this approach by a budget-based methods which keeps a tally of gains and losses as requests are processed. We also analyze systems with infinite states and devise several strategies to transition between states. The thesis gives results both in terms of theoretical analysis as well as a result of extensive simulation.

Acknowledgements

I would like to thank my advisor, Dr. Wolfgang Bein with all of his mentorship throughout this entire process. His dedication, guidance, and friendly demeanor inspired me to strive ahead in my research and also inspired me to grow as a person in the profession of research and academia. His willingness to always give me advice in various stages of my research and my academic career will not be forgotten and I will always be grateful. He would give me guidance on how to improve my skills in my profession as a researcher and teaching. We traveled to Tokyo, Japan twice and both times he took me under his wing, and made a stressful situation of traveling to a foreign country (since it was my first time traveling to a foreign country) into a memorable experience of a lifetime. I am truly grateful that he was my advisor. He was more than just an mentor to me, he was always a good friend.

I also acknowledge support under National Science Foundation Grant IIA 142784. The grant allowed me to visit the University of Electro-Communications to work on this topic. While in Japan, I had the opportunity to meet Dr. Hiro Ito and Dr. Jun Kawahara who worked and aided me in my work, which I am very thankful. I would also like to thank Dr. Lawrence Larmore. While being his graduate teaching assistant for several semesters, he opened my mind and allowed to to think like an academic professional which helped me in my work and with my teaching. I would also like to express my gratitude to Dr. Laxmi Gewali and Dr. Ajoy Datta for giving me the opportunity to work here at this university which further increased my development in the academic profession and giving me a chance to proof myself when not many did. The entire staff here created

an environment for me which felt like a second home with a second family, and my mentality was never "I have to go to work today", but rather "I get to go to work today." I would like to thank my entire committee, Dr. Wolfgang Bein, Dr. Robert Boehm, Dr. Ajoy Datta, Dr. Lawrence Larmore, and Dr. Andreas Stefik for their discussions and advice throughout the process. Also discussions with Rüdiger Reischuk of Universität Lübeck during his sabbatical visit are acknowledged.

I would like to thank my mother and father for creating an atmosphere at home which allowed me to focus on my goals. My mother worked two jobs to provide for me and my father was always there for me whenever things did not always look positive. Lastly, I would like to thank everyone who believed in me throughout this long journey.

James Andro-Vasko

University of Nevada, Las Vegas

December 2017

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Power Down Problem

Consider a machine with an ON state, an OFF state, and possibly a set of intermediate power states. Each state has an idle cost, or unit cost, which is the cost to remain idle in that particular state, and each state also has a power up cost which is the cost to switch to the ON state. The machine must be in the ON state in order to process a request so when a request arrives, the machine is either in the ON state and will process the request or it needs to power up to the ON state, if the machine was in any other state. The machine can switch to a lower power state at any time but will incur a power up cost a request arrives after the machine switched to the lower power state.

Given the set of requests, given their release times and deadlines, the goal is to minimize the power consumption needed to process all of the requests. It is quite trivial to compute the optimal power usage given a set of requests, we would determine the idle time between two requests, and then we would choose the state that minimizes the idle and power up cost in the duration. However, for the power down problem, we will consider the online model, where we do not know when the idle period will end and we must decide which state to use at the current point in time.

1.2 Online Algorithms

When we develop an online algorithm, we develop an algorithm that makes its decisions without knowing any of the future input. Our goal is to obtain an online algorithm that minimizes its maximal cost for all inputs. We use the term competitive ratio which is defined to be $Cost_A(\sigma) \leq c \cdot Cost_{opt}(\sigma)$ where $Cost_A(\sigma)$ is the cost of an online algorithm with input sequence σ and $Cost_{opt}(\sigma)$ is the cost of the optimal offline algorithm for input sequence σ, if the inequality holds for any input sequence,

then we say that the online algorithm is c-competitive which are discussed in [5, 21, 31, 47]. Other areas of online algorithms that have been researched is the online paging problem [19, 44, 49, 50], mostly the LRU in the online setting has been studied [15, 18, 27, 46]. The k server problem has been in [16, 23] has been studied where a a request arrives in a grid and servers are moved to the location. There has also been work done in online binpacking [40]. The input sequence is given to the online and offline algorithm by an adversary such that the competitive ratio is maximized [16, 21]. For a given online problem, we have a set of online algorithms A_i, the offline algorithm OPT, and the input σ_i for each algorithm A_i and we choose A_i such that has the smallest competitive ratio for all of the online algorithms and this algorithm is the best algorithm.

We use the online model for the power down problem because in real world applications, the input sequence will not always be known. Queuing theory can also be applied to the online power down problem and also assumptions can be made in practice about when the machine will be more active and when it will be less active. For example with power plants and power grids, usually at night the demand for power is lower than during the day, and during different seasons of the year, the power demand changes and scheduling whether to use a full scale power plant or a power grid can be trivially done. Also, when a new item appears on a streaming service of any kind, it can be assumed the severs will take a big hit in its work load so it can also easily be determined that the machine will be very busy and power down strategies do not play such a significant role. However, if it cannot be predicted how busy or idle the machine will be is where online competitive analysis does become important. The competitive ratio for an online algorithm acts as a guarantee that insures that we can not do any worse than the upper bound of the algorithm. Suppose we have an algorithm $\mathcal{A}$ that has better runtime, or smaller cost, than algorithm $\mathcal{B}$, but in the worst case $\mathcal{A}$ performs much worse than $\mathcal{B}$, then the better choice would be to use algorithm $\mathcal{B}$ for our problem.

Usually we choose the algorithm with the lower competitive ratio because we want a more favorable result even in the worst possible case, unless the algorithm with a worse competitive ratio is larger only by an arbitrarily small amount and performs much better in the average and in the best case. In this case, we would have to do careful and extensive analysis by applying the algorithms against all types of input in relatively large quantity of inputs for each test, in order to make an assumption that the algorithm with a slightly less competitive ratio has better results in the best and average case. Even in this scenario, we still want to choose the algorithm with the best competitive ratio or has a competitive ratio that is only slightly worse than the algorithm with the best known competitive ratio.

1.3 Prior Work in Green Computing and Applications of Green Computing

In 2013, 91 billion kilowatt-hours of energy has been used up by U.S. data centers, so research in the area of green computing has a significant role [24]. In fact according to Google, energy costs are often larger than hardware cost and ways to minimize energy consumption are crucial [13]. In green computing, there has been a great deal of work done in the area of speed scaling [29, 42, 51]. In [11, 12], a new lower and upper bound is introduced which are $e^{\alpha-1}/\alpha$ and $2e^{\alpha+1}$, where α is some constant used to compute the power used when the CPU is scaled up or down to complete a set of jobs, which is an improvement from the previous competitive ratio of 27 to 6.7 when $\alpha = 3$. In [28], the SOA algorithm is introduced for the speed scaling problem which is 4-competitive for throughput and $(\alpha^\alpha + \alpha^2 4^\alpha + 2)$-competitive. Competitive snooping [38] where each processor keeps track of which blocks of data to retain to drop to have minimal communication on the system bus which reduces energy cost. Other research has been done with power down problems over a network to reduce energy cost of idling server machines while maintaining an effective network [3, 4, 35, 45].

Some applications of green computing is used on power grids [20, 26] which are small scale power plants that power on when there is low demand and when power grids are active, the larger scale power plants can power down. In multiprocessors or thread environments, processes at times have to wait to enter a critical section in which they will either be put to sleep or they will spinlock [37] in which spinlocking costs power and going to sleep uses no power but then there is a power up cost, directly related to the power down problem. Similar issues occur in network between sending bursts of packets whether to keep the connection open or closed [36, 41]. With cache coherency, there are times when the data and cache are updated and there are strategies as to when to update or invalidate that data in main memory [7, 25], performing several updates separately can be costly, so determining the right moment to update to minimize cost can be related to the power down problem as to when to power down after being idle for a given amount of time. Other work that has been done is on the online capital investment problem which handles special case for power down problem in which the power up costs are equal for all states $4 + 2\sqrt{2}$ [10]. Damaschke et al [22] improves this special case to 4-competitive. We will focus our attention on the online power down problems in which randomization will not be applied.

The power down problem is relate-able to speed scaling because when a machine is running a processes it can adjust the speed of the processor which acts as some power state. Our focus will be on the power down problems in the online setting, where we have a machine with several power states, and we transition to lower power states as the machine is idling to save power. Some previous

work in the area of power down has been the 2 state power down problem and the discrete multi state power down problems [6, 34, 47] in which there exists an algorithm that yields the optimal competitive ratio which takes $O(n^2 \log n \log(1/\epsilon))$ time where n is the number of states and ϵ is an approximation value [9]. There also has been work done using randomized algorithms in the power down problems which yielded a competitive ratio of $e/e-1$ [32, 33, 39].

1.4 Contributions

In this paper, we do extensive work on the power down problem with our main focus on power state machines with few states. In practice, most machines have up to 3 or 5 power states, and this allows us to concentrate on this smaller problem which also simplifies the strategy in devising a schedule that yields the optimal online cost. In Augustine et al.[9], an algorithm is introduced that obtains the best switch times to get the minimal competitive ratio within an ϵ approximation, for an n state machine. Our approach on the 3 state machine, introduced in chapter 4, we have a mechanism of obtaining the exact minimal competitive ratio by computing the switch times that yield the best competitive ratio in constant runtime.

For the 5 state machine, in chapter 5, we show our approach to obtain the switch times to obtain the minimal competitive ratio within an ϵ, which is also a simpler approach from the known algorithm for n state machine algorithm, this algorithm was inspired using a similar technique of using a binary search on the range of possible competitive ratios as done in [9], however, once again we compute the switch times in constant time without applying any other search, which simplifies and speeds up the problem. We then discuss the continuous state problem, in chapter 6, where we transition to lower power states using a continuous curve for the unit and power up costs, where we do not have discrete time units for transitions, we use another curve that dictates how we switch to lower cost states throughout the idle period. We can imagine the continuous power down problem to be an analog throttle control where we can adjust the dial rapidly or slowly. We show strategies that show how rapidly we switch to a lower power state and summarize the costs and competitive ratios of using a set of strategies.

After we set the foundation power state machines for various amounts of states, we analyze tapering down strategies. In chapter 7, we introduce the decrease and reset algorithm (DRA) for a two state system, which adjusts the delay times according to previous delay times we had before requests. The concept of any tapering based approach is that we decrease the wait times that the machine stays in on state while idling. In chapter 2, we show a proof that states that there is a single instance that the two state machine powers down to obtain the minimal competitive ratio. When we apply this tapering approach, since the idle time is decreased, the competitive ratio increases by

some small arbitrary amount. In a competitive analysis sense, tapering down leads to an algorithm that yields are larger cost, however in practice, if we have a machine that spends a great deal of time idling and powering down between requests, we can save energy if we decide to power down to the off state earlier than we normally would have. However, if the system becomes active, then we reset the idle time to a larger amount to decrease the energy used if we power down too soon. Also, in chapter 7, we introduce a budget based tapering down approach that uses some variable which will denote the energy saved, which we call a budget, to adjust the wait time between requests. This approach is similar conceptually to the DRA in terms of how the machine tapers down, but the adjusted wait times in the budget based approach attempts to calculate the switch time to be more cost efficient in the worst case than the DRA. We compare the results of these two approaches in chapter 10 for a set of input, which are a set of requests, to show the costs of the two algorithms when requests are arriving soon after one another or when the requests arrive distantly from each other.

In chapter 8, we further analyze the decrease and reset algorithm and apply it to the three state model from chapter 4, to attempt to reduce the wait time in the high power on state and the wait time in the intermediate state. Similar to the two state DRA, when a request arrives after the machine powers down, the idle time is decreased, however we witness an odd behavior that the on state idle duration decreases and the idle time in the intermediate state remains unchanged between requests that arrive distant from each other. The intermediate state idle duration starts decreasing only when the on state duration is depleted to a wait time of 0, and the intermediate state tapers down at a faster rate than when the on state is tapering down its wait time.

In chapter 9, we introduce a budget based technique as we did in chapter 7, but with three states. The behavior differs from the three state DRA by the way the idle times taper down. In the budget based model, the on state and the intermediate state wait time are tapering down simultaneously because we apply the energy gained to compute a new wait time for both the on and the intermediate states. In chapters 10 and 11 we compare the DRA with the budget based for two states and three states with a set of various inputs which are generated randomly to further analyze the costs of the two techniques against each other as well as with the optimal offline cost and online algorithm that does not use any tapering which is the optimal online algorithm.

Chapter 2

Two State Problem

We first consider systems with two states. The two state problem is a similar to a well known problem: the ski-rental problem . We give a number of simple results which have been published before, but which are implicit in the solutions to the ski-rental problem. There are two states ,ON and OFF; in the ON state, there is a standby cost α and there is no cost to power up when a request arrives (since we are already in the ON state). If the machine is in the OFF state when the request arrives, it has to power up to the ON state in order to processes the request. There is a power up cost β in the OFF state and there is no standby cost. The goal is to determine the time to switch to the OFF state when the machine is idle, to minimize the power consumption of the machine in the worst case. For all of the instances of the power down problems, we have an input sequence σ which is a sequence of job requests for the machine.

$$\sigma = ((r_1^s, r_1^e), (r_2^s, r_2^e), (r_3^s, r_3^e), ..., (r_n^s, r_n^e))$$

Each pair (r_i^s, r_i^e) denotes the start and end times of job i respectively. It is clear that the length of job i will be $r_i^e - r_i^s$, which will simply be the difference of end time and start time for job i. The machine uses power when processing a request and when the machine is idle, since our focus with the power down problem is conserving energy when the machine is idle, we do not consider the cost to process an actual request. Therefore we can collapse r_i^s and r_i^e into a single time instance, σ can be rewritten to:

$$\sigma = (r_1, r_2, r_3, ..., r_n)$$

So the issue is when to power down after any request in the sequence σ. A few obvious methodologies are: to always power down after each request or always remain in the ON state. A more complex methodology is to wait for a duration in the ON state and eventually power down after some time. In the always OFF strategy, may not always be a bad strategy if the machine is rarely

used, if there is a long idle duration between requests. Once again, the adversary will maximize the cost of this algorithm. The adversary will send the requests with arbitrarily small delay times, which will incur an unnecessary power up cost after each request, since the power up cost outweighs the idle cost between the requests. The cost of the online algorithm will be β for each request since it has to power up each time, and the machine will never remain idle in the ON state so there will be no standby cost incurred. The offline algorithm will once again have a nonexistent cost since it is in ON state at the start of each request and since the machine must be in the ON state to process the request, there is no idle cost and no power up cost.

Lemma 2.0.1. *The always OFF strategy has an unbounded competitive ratio*

Proof. For the sake of competitive analysis, we assign an initial cost to the offline algorithm β which is the initial power up cost, the competitive ratio can be seen below

$$\text{Competitive Ratio} = \frac{n\beta}{\beta}$$

In the sequence, there will be arbitrarily many requests, so $n \to \infty$, the offline cost does not incur any additional cost since it does not power down after each request but rather stay ON and process each subsequent request. The online strategy will power down and up after each request which incurs the the power up cost each time which causes this strategy to be unbounded.

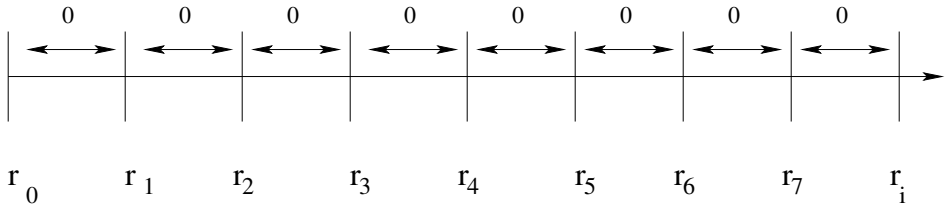

Figure 2.1: Cost of always OFF approach

□

In the always remain ON approach, the machine never powers off regardless of the length of its idle period. This approach in a practical sense might not be a bad idea if requests are arriving after each other with high frequency. If we consider the adversary, the next request arrives after an arbitrarily long time which will maximize the cost of the online algorithm, which causes the energy consumption to grow continuously. The online algorithm could have saved power if it had powered down at some point rather than staying in the ON state throughout the duration in an idle state. Meanwhile the optimal offline algorithm will just power down after the request since the offline algorithm knows the next request will arrives after a long period of time

Lemma 2.0.2. *The always ON strategy has an unbounded competitive ratio*

Proof. The cost of the online algorithm is αt where t will be the idle length between the two requests. The offline cost is β due to its initial power up and the machine powers down right after the request and does not power up due to the fact that the next request arrives after a long period of time. .

$$\text{Competitive Ratio} = \frac{\alpha t}{\beta}$$

We can see the the offline cost does not grow while the online cost is growing at a linear rate which makes the competitive ratio unbounded.

Figure 2.2: Cost of always ON approach

□

Another option is to remain idle in between requests for some time and eventually power down if the machine waits long enough without a request arriving. The issue is determining how long the idle duration will be to minimize the online algorithm cost against the adversary. Several papers introduced this problem which is equivalent to the ski rental algorithm which can be found in [6, 34, 43, 48, 17, 31, 47]. The idea is if one wishes to go skiing and this individual needs skis, either they can be bought or rented.

There is a cost to rent and buy the skis and renting will typically cost less than buying. It is clear that if one goes skiing rarely, then there is no point buying and thus renting would the favorable option, and if one goes skiing frequently, then buying would be the favorable option. For the ski rental problem, if one rents the skis one is guaranteed to go skiing once again, and once one buys the skis, one does not go skiing again, this scenario is produced by the adversary. This model can be directly applied to the 2 state power down problem, where remaining idle is equivalent to renting skis and powering down is equivalent to buying the skis, we can amortize the power down cost because the machine will most likely be used again so there will be a cost to power up.

Call the offline algorithm OPT and the online algorithm A. The cost of the offline algorithm is $OPT = \min\{\alpha t, \beta\}$. For A, it stays idle in the ON state and switch to the OFF state after time t, the adversary guarantees the request arrives after the machine powers down; the cost of A in the worst case is $\alpha t + \beta$. From the cost of OPT, we can see that its cost does not exceed β for any

request, $t \leq \beta/\alpha$. For the cost of A and OPT, we can compute the competitive ratio for any t value. We will use values $\alpha = \beta = 1$.

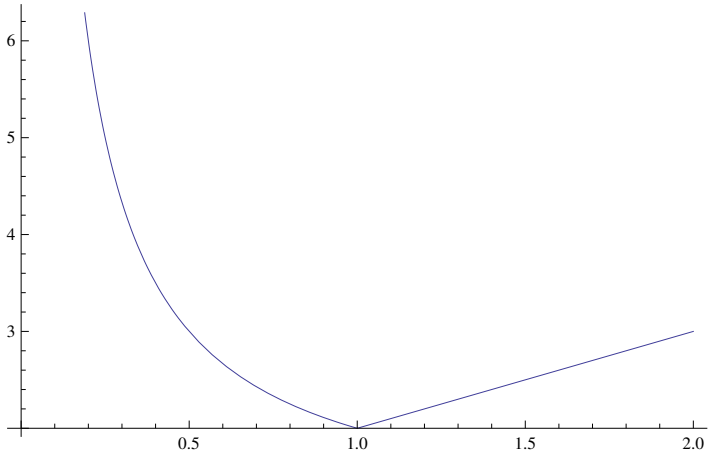

Figure 2.3: Competitive ratio for various delay times

Lemma 2.0.3. *The competitive ratio is minimized when $t = \beta/\alpha$*

Proof. We refer to Figure 2.3, $\beta = r = 1$ and so $\beta/\alpha = 1$. The graph shows that when the delay time is β/α, the competitive ratio is minimized. Let us assume that the competitive ratio is not minimized when $t = \beta/\alpha$. First it will be assumed that when $t > \beta/\alpha$, the competitive ratio is minimized.

$$\text{Competitive ratio} = \frac{\alpha t + \beta}{\beta} = 1 + \frac{\alpha t}{\beta}$$

As t becomes larger, the competitive ratio also will grow larger. So the competitive ratio is not minimized if $t > \beta/\alpha$. Now let us assume that the competitive ratio is minimized when $t < \beta/\alpha$.

$$\text{Competitive ratio} = \frac{\alpha t + \beta}{\alpha t} = 1 + \frac{\beta}{\alpha t}$$

One can see that as t gets smaller, the competitive ratio will grow. The competitive ratio is not minimal when $t < \beta/\alpha$. Therefore, the competitive ratio can only be minimized when $t = \beta/\alpha$. □

Theorem 2.0.4. *The two state power down problem is 2-competitive.*

Proof. From lemma 2.0.3, we know that the competitive ratio is minimized when $t = \beta/\alpha$. We have the following competitive ratio:

$$\text{Competitive ratio} = \frac{\alpha t + \beta}{\beta} = \frac{\beta + \beta}{\beta} = 2$$

If we were to compute the competitive ratios for idle times $t < \beta/\alpha$ and $t > \beta/\alpha$, the competitive ratios will be larger than 2 in either case. For the reason that there is an adversary, there is no way to improve this upper bound which is 2-competitive [6, 31, 34, 47]. □

Chapter 3

Multiple State Problem

In the multiple state problem we are given an ON and OFF state with a number of intermediate lower power states. State 0, S_0 will denote the ON state and S_n will denote the OFF state. Multiple power states has been trending lately, we can see the power specs for Windows [1] and Apple [2]. Windows machines have 7 power states: ON, OFF, 3 intermediate sleep states, hibernate and soft off state. For each state S_i, there will be an idle cost to remain in that state and a cost to power up to the ON state. The cost each state will incur will be $Cost(S_i(t)) = \alpha_i t + \beta_i$ where t will denote the duration in state i. State 0 will be the ON state so $\beta_0 = 0$ and state n will denote the OFF state so $\alpha_n = 0$. The idle costs will satisfy the following sequence $\alpha_0 > \alpha_1 > \alpha_2 > ... > \alpha_n$ and the power up costs will satisfy the following $\beta_0 < \beta_1 < \beta_2 < ... < \beta_n$.

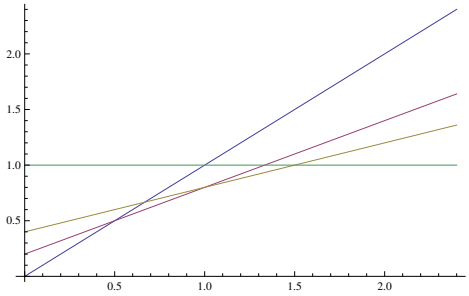

Figure 3.1: Cost incurred by each state S_i

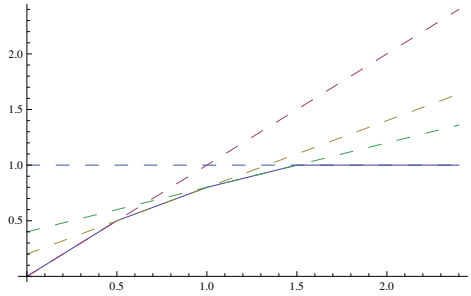

Figure 3.2: Optimal cost for multiple states

Figure 3.1 is a plot of each $Cost(S_i)$, using this figure, figure 3.2 shows the cost of the offline algorithm which is denoted by the solid line. The cost curves in this example are arbitrary. Each cost curve for each state can be plotted and the intersection between two functions determine which state the offline algorithm will use given the length of the idle time between requests. The cost of

the offline algorithm will be $Cost(S_j) = \min_{i=0}^{n}\{\alpha_j t + \beta_j\}$ where t is the delay time. Let state i be the highest power state before state j such that $Cost(S_i) = Cost(S_j)$.

$$t_j = \frac{\beta_i - \beta_j}{\alpha_j - \alpha_i} \tag{3.1}$$

The online algorithm will also use these t_j times to denoted which state will be used. Unlike the offline algorithm, it is not known when the request arrives, so it cannot choose which state to start at the beginning of the idle period but rather start from the ON state and remain in that state for t_i $(i > 0)$ time units and at that time will switch to state i and will repeat this process until the machine will power down. This algorithm is called the lower envelope algorithm (LEA) which was introduced in Irani et al [30].

Lemma 3.0.1. *For any system, the worst case competitive ratio occurs at a transition time [9].*

Proof. Let $A(t)$ be the cost of an online algorithm at time t and $OPT(t)$ be the cost of an offline algorithm at time t and ρ denotes the competitive ratio. The earliest time that $A(t) = \rho OPT(t)$ is denoted by $\bar{t}$. We assume that $\bar{t}$ is not a switch time to a lower state for A. So for some δ, the interval $(\bar{t} - \delta, \bar{t} + \delta)$ is increasing at a linear rate since there is no transition to a lower state in this interval. If the interval $(\bar{t} - \delta)$ is strictly less than the competitive ratio and $A(\bar{t}) = \rho OPT(\bar{t})$, then during this interval, the slope of the online algorithm is greater than the slope of the offline algorithm. That means at $t > \bar{t}$, $A(t) > \rho OPT(t)$ which leads to a contradiction. If the competitive ratio is equal for the duration $(\Delta t - \delta)$ and $A(\bar{t}) = \rho OPT(\bar{t})$, the $A(t \bar{+} \delta) = \rho OPT(t \bar{+} \delta)$ holds but the eventually the online algorithm will incur a power up cost (due to the adversary) and the offline algorithm will not incur this power up cost so the competitive ratio increases and thus the competitive ratio was not maximized at time $\bar{t}$ which also leads to a contradiction. □

In LEA, we can see that after every transition time, the cost jumps by β_j at t_i (where state j is a lower power state than state i) when the machine powers down to state i. And the competitive ratios will reflect those jumps in the following figures.

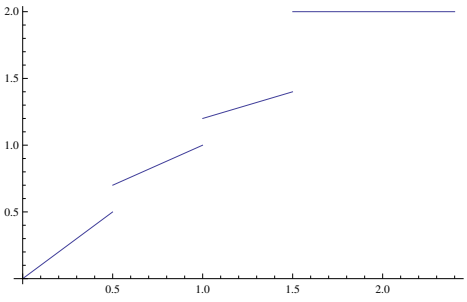

Figure 3.3: Cost of LEA

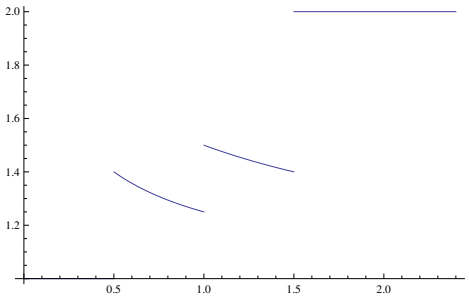

Figure 3.4: Competitive ratio of LEA

It can be seen that in this instance the competitive ratio is less than 2 for the entire idle duration except when the machine powers down to the OFF state. Once again the online costs and the competitive ratios in figures 3.3 and 3.4 use arbitrary standby and power up costs.

Theorem 3.0.2. *LEA is 2-competitive [30].*

Proof. The competitive ratio for LEA is:

$$\mathcal{R}_{\mathcal{LEA}} = \frac{\sum_{i=0}^{j-1}\{\alpha_i(t_{i+1}-t_i)\}+\beta_j}{\alpha_k t_j+\beta_j} = 1 + \frac{\sum_{i=0}^{j-1}\{\alpha_i(t_{i+1}-t_i)\}-\alpha_j t_j}{\alpha_k t_j+\beta_j}$$

In order to show that LEA is 2-competitive in the worst case, we show the following inequality

$$\sum_{i=0}^{j-1}\{\alpha_i(t_{i+1}-t_i)\}-\alpha_j t_j \leq \alpha_k t_j+\beta_j$$

Using (3.1), we can rewrite the above inequality

$$(\beta_1-\beta_0)+(\beta_2-\beta_1)+...+(\beta_j-\beta_{j-1})-\alpha_0 t_0 \leq \alpha_k t_j+\beta_j$$

$$\beta_j-\beta_0-\alpha_0 t_0 \leq \alpha_j t_j+\beta_j$$

The values of t_0 and β_0 are both 0. So the inequality will always hold for every j. If $j=n$ then we have $\beta_n \leq \beta_n$ and then the competitive ratio will be 2. Which is depicted in figure 3.4. □

Theorem 3.0.3. *There is a $(3+2\sqrt{2})$-competitive strategy for any system [9].*

Proof. The term $d_{i,j} = \beta_j - \beta_i$ is introduced and therefore $\sum_{i=1}^{k} d_{i-1,i} = \beta_k$. This allows us to amortize the power up costs to pay a cost each time the machine switches to a lower power state and not pay for the power up cost since that cost has already been incurred. Now we assume that for γ, $\beta_i \geq \gamma\beta_{i-1}$ for all i. The online cost is $A(t) = \sum_{j=0}^{i-1}(\alpha_j(t_{j+1}-t_j)+d_{j,j+1})+\alpha_i(t-t_i)$ using LEA. The offline algorithm will be $OPT(t) = \sum_{j=0}^{i-1}\alpha_j(t_{j+1}-t_j)$. If a new request occurs at a switch time t_i, the we have the following online cost

$$A(t_i) = \sum_{j=0}^{i-1}(\alpha_j(t_{j+1}-t_j)+d_{j,j+1})$$

We obtain a slightly larger upper bound since $\sum_{j=0}^{i-1} \beta_j > \sum_{j=0}^{i-1} d_{j,j+1}$

$$A(t_i) \leq \sum_{j=0}^{i-1} \alpha_j (t_{j+1} - t_j) + \sum_{j=0}^{i-1} \beta_j$$

Since $\beta_i \geq \gamma \beta_{i-1}$, we say that $\beta_i/\gamma \geq \beta_{i-1}$ and $\beta_i/\gamma^2 \geq \beta_{i-2}$, and we generalize this to obtain $\beta_i \sum_{j=0}^{i-1} \gamma^{-(i-j)}$

$$\leq OPT(t_i) + \beta_i \sum_{j=0}^{i-1} \gamma^{-(i-j)}$$

Since $\beta_i < OPT(t_i)$

$$\leq (1 + \frac{\gamma}{\gamma - 1}) OPT(t_i) = \frac{2\gamma - 1}{\gamma - 1} OPT(t_i)$$

Given some γ for all t we have a system that is $\frac{2\gamma-1}{\gamma-1}$ competitive. Now we assume that $\beta_i \geq \gamma \beta_{i-1}$ does not hold. In this case, the cost of $OPT(t) = \sum_{j=0}^{i-1} \alpha_j (t_{j+1} - t_j)$ will not be optimal since not all the states will be used in the optimal solution because of the fact that $\beta_i \geq \gamma \beta_{i-1}$ does not always hold. We will consider an alternate optimal cost denoted by $OPT'(t)$. We will have a set of states $\mathcal{S}$ that OPT' uses. We first start with $\mathcal{S} = \{S_n\}$, because $\mathcal{S}$ must contain the OFF state. We will look at the states in reverse order, initially S_n is added to $\mathcal{S}$, now we seek to find a state $i < n$ such that $\gamma \beta_i \leq \beta_n$. Now that state i is added to $\mathcal{S}$, we find the largest j such that $0 \leq j < i$ and $\gamma \beta_j \leq \beta_i$. And now we apply the same offline algorithm on the set $\mathcal{S}$ to obtain a different offline cost $OPT'(t)$.

Suppose there are states S_i, S_l, and S_k where $i < l < k$ and $S_l \notin \mathcal{S}$. We have $OPT(t) = OPT'(t)$ in intervals $t \in [t_i, t_{i+1})$ and $t \in [t_j, t_{j+1})$. The cost of $OPT'(t)$ for $t \in [t_l, t_{l+1})$ will be $\min\{\alpha_i t + \beta_i, \alpha_j t + \beta_j\}$ and the cost for $OPT(t) = \alpha_l t + \beta_l$. $OPT'(t)$ will choose $\alpha_i t + \beta_i$ as its min value and since $S_l \notin \mathcal{S}$ then $\gamma \beta_l > \beta_i$ and $\alpha_l > \alpha_i$.

$$OPT'(t) = \beta_i + \alpha_i t \leq \gamma(\alpha_l t + \beta_l) = \gamma OPT(t)$$

Which shows that OPT is larger than OPT' by a factor of γ which means $A(t) \leq \frac{2\gamma-1}{\gamma-1} OPT'(t) \leq \gamma \frac{2\gamma-1}{\gamma-1} OPT(t)$. The expression $\gamma \frac{2\gamma-1}{\gamma-1}$ is minimal when $\gamma = 1 + \frac{1}{\sqrt{2}}$ which means we get a competitive ratio of $3 + 2\sqrt{2}$. □

Let A be a ρ-competitive strategy, then there exists an algorithm A' such that it is ρ eager which is also ρ-competitive [9]. A ρ eager strategy is defined to be when the machine transitions to a lower power state at time t, such that $A'(t) = \rho OPT(t)$. We can always choose transition times such that at a transition time t, $A'(t) < \rho OPT(t)$. To show that a ρ eager strategy does exist, we will assume $\mathcal{T}$ is the earliest transition time that is not eager. We will let $\mathcal{T}' < \mathcal{T}$ be an earlier transition time, if there is no earlier transition time then $\mathcal{T}$ is the earliest transition time so $\mathcal{T}' = 0$.

The cost of A in the interval $(\mathcal{T}', \mathcal{T})$ has no transition so it is continuous over that interval. Let $\bar{\mathcal{T}}$ be the earliest time after $\mathcal{T}'$ such that $A(t) = \rho OPT(t)$. Consider an online algorithm A' which is identical to A except it transitions from S_i to S_j at time $\bar{\mathcal{T}}$ instead of $\mathcal{T}$. So in the interval $[\mathcal{T}, \bar{\mathcal{T}})$, both A and A' will have the same cost. Both A and A' will be ρ-competitive at time $\bar{\mathcal{T}}$ however, since A' transitions to a lower power state at $\bar{\mathcal{T}}$, at time $\mathcal{T}$, $A'(t) < A(t)$ since A' will use less power while in S_j and have a smaller slope in the interval $[\bar{\mathcal{T}}, \mathcal{T}]$ and A will have to pay the same cost $d_{i,j}$, therefore A' will also be ρ-competitive. This process can be repeated for all states in the system to obtain a ρ-competitive strategy that is also ρ eager.

Theorem 3.0.4. *Given n states, there is a strategy that computes a schedule that obtains the optimal competitive ratio in $O(n^2 \log n \log(1/\epsilon))$ [9]*

Proof. Consider a function $f(t) = \alpha_i(t - t_i) + \rho OPT(t_i) + d_{i,j}$ which gives the cost of the online algorithm once it enters state i. We attempt to find a switching time t from state i to state j while remaining ρ eager, if there is no such t then there does not exist a ρ eager strategy from state i to j. We compare the cost of function $f(t)$ with $\rho OPT(t)$, we will let OPT be in state s_l at time t_i. We will denote b_0 the time which OPT is in state s_0, b_1 in state b_1, and b_n is in state s_n. We have a range $[b_l, b_n]$, in which the time t such that $f(t) = \rho OPT(t)$ must exist in this range. We can perform a binary search to determine the smallest possible range in which t could exist. Starting from s_l and s_n as the endpoints for OPT, we choose time b_{mid} and if $f(b_{mid}) > \rho OPT(b_{mid})$ then we update the right endpoint to this b_{mid} value and update the left endpoint s_l to this midpoint, due to the fact that $f(t)$ is a linearly increasing function and ρOPT is a concave function. Once this binary search completes, we know which state ρOPT will use so then we can determine a t value such that $f(t) = \rho OPT(t)$, which will be the time in which the online algorithm switches from state i to state j while being ρ eager. This process will take $O(\log n)$ in which n is the number of states.

We must perform the operation between all pairs of states i and j, which will be $O(n^2)$ possible pairs, and finding the ρ eager transitions for all the states will take $O(n^2 \log n)$. Once we compute the t values for all i and j values we perform a search that finds a path from s_0 to s_n using the transition times between all states i and j if there exists such a schedule and an error if one does not exist. This search does not increase the asymptotic complexity so for each ρ that is given, a schedule or an error will be returned in $O(n^2 \log n)$ time.

From theorem 3.0.3, we know that the optimal competitive ratio must be in the range $[1, 3 + 2\sqrt{2}]$. The cost for each state is shown below. Let ρ^* denote the optimal competitive ratio and $\rho < \rho^* + \epsilon$, and thus ρ will be an approximation. Now we will perform the $O(n^2 \log n)$ algorithm for each value of ρ in the range $[1, 3 + 2\sqrt{2}]$. We will once again perform a binary search in that range to find the optimal ρ. Using the ϵ approximation value we have to perform the $O(n^2 \log n)$ operation

$\log[(3+2\sqrt{2}+1/2)(1/\epsilon)]$ number of times. We use inverse ϵ because if ϵ is arbitrarily smaller, the number of steps increase, and then we have a more accurate ρ value, and takes less number of steps if ϵ is arbitrarily larger. This binary search will find the nearest optimal ρ value depending on ϵ. So the entire process takes $O(n^2 \log n \log(1/\epsilon))$ time. □

In the rest of this dissertation, we work on more specific systems that have small number of states. This is designed to isolate the problem and perform a dense analysis and obtain more exact schedules and apply power management strategies onto these systems, and we also show how transition times are computed and how they differ depending on the costs of the ON, OFF, and any intermediate states. This chapter was a survey of a previously solved problem of a machine with n states and we branch off to work on system with small states as well as a continuous state system, and we will apply tapering strategies onto these systems.

Chapter 4

Three State Problem

In this chapter, we give a complete characterization of the three state system. In the three state problem, the states will be ON, OFF, and INT where INT will be some lower power intermediate state. As discussed earlier, the offline algorithm will schedule itself to yield the optimal cost. We seek to choose an online algorithm that has the best competitive ratio among all possible online algorithms. The online algorithm will have two switching times, one for when there is a switch from ON to INT and one for when there is a switch from INT to OFF.

State	Idle Cost	Power Up Cost
ON	1	0
INT	$a \in (0,1)$	$d \in (0,1)$
OFF	0	1

Table 4.1: Three state costs

In the online model, the machine will have switching times x_1 and x_2 which denote the switch times from ON to INT and INT to OFF respectively. The offline model will have x_{opt_1} and x_{opt_2}, for these values, the optimal offline algorithm will decide from which state it will begin based on the idle time, if the request arrives before x_{opt_1} then it will be in the ON state during the idle duration, if the request arrives between x_{opt_1} and x_{opt_2} then it will be in the INT state, and if the request arrives after x_{opt_2}, then it will be in the OFF state. We will have Cost_{opt} defined in the following

way:

$$Cost_{opt}(t) = \begin{cases} t & \text{if } t < x_{opt_1} \\ at + d & \text{if } x_{opt_1} \leq t < x_{opt_2} \\ 1 & \text{if } t \geq x_{opt_2} \end{cases}$$

Since this is an offline model, the values of x_{opt_1} and x_{opt_2} are known in advance given the values for a and d.

Lemma 4.0.1. *For the optimal offline model,* $x_{opt_1} = d/(1-a)$ *and* $x_{opt_2} = (1-d)/a$

Proof. The offline cost curves are $f(t) = t$, $f(t) = at + d$, and $f(t) = 1$. The curve $f(t) = t$ intersects with $f(t) = at + d$ when $t = d/(1-a)$, before this time, the ON state yields the optimal cost and after this time, the INT state yields the optimal cost. The curve $f(t) = at + d$ intersects with $f(t) = 1$ at $t = (1-d)/a$. If the request arrives before $t = d/(1-a)$, then from 0 to $d/(1-a)$ the optimal cost is obtained using the cost curve $f(t) = t$ which is the ON state, if the request arrives between $d/(1-a)$ to $(1-d)/a$, then the optimal cost is obtained using the cost curve $f(t) = at + d$, which is the INT state, and if the request arrives at or after $(1-d)/a$, the optimal cost curve is $f(t) = 1$ which is the OFF state. Therefore $x_{opt_1} = d/(1-a)$ and $x_{opt_2} = (1-d)/a$. □

Using the formulas from lemma 4.0.1, we can compute the x_{opt_1} and x_{opt_2} and hence compute the offline costs. We can see the curves that represent the offline costs below.

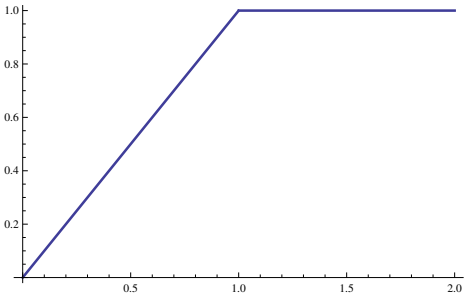

Figure 4.1: Optimal cost when state INT has costs a = 0.6 d = 0.4

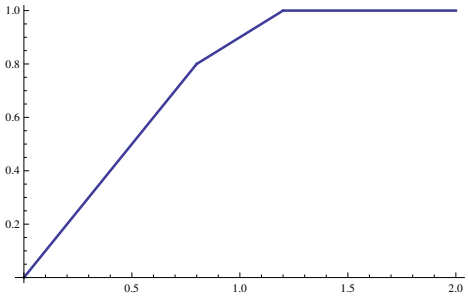

Figure 4.2: Optimal cost when state INT has costs a = 0.5 d = 0.4

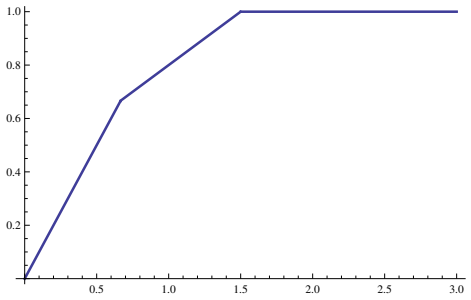

Figure 4.3: Optimal cost when state INT has costs a = 0.4 d = 0.4

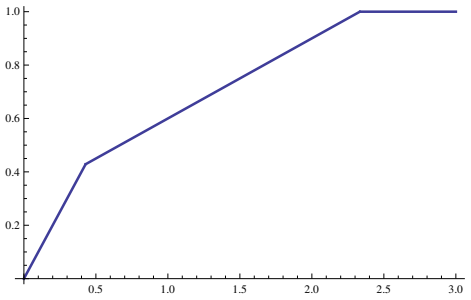

Figure 4.4: Optimal cost when state INT has costs a = 0.3 d = 0.3

We see that in figure 4.1, the offline algorithm goes from the ON state directly to the OFF state, in the other cases we see that there are periods where the INT state is utilized.

Lemma 4.0.2. *If $a + d \geq 1$, the offline algorithm will not use the intermediate state.*

Proof. We assume that if $a+d \geq 1$, the offline algorithm will use the intermediate state. Once again we have the three curves $f(t) = t$, $f(t) = at + d$, and $f(t) = 1$. Figure 4.5 and 4.5 shows the two possible cases when $a + d = 1$ and when $a + d > 1$.

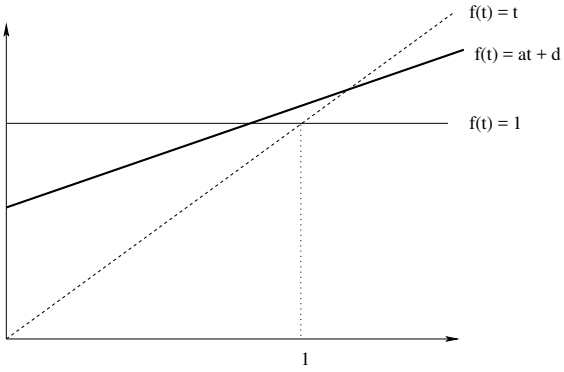

Figure 4.5: ON, INT, and OFF state curves when $a + d > 1$

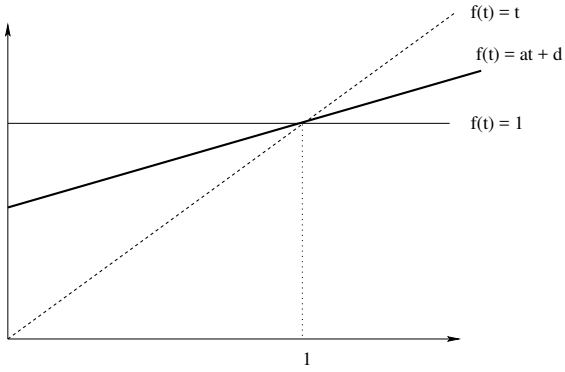

Figure 4.6: ON, INT, and OFF state curves when $a + d = 1$

If $a + d > 1$, then at time 1, the cost of the INT state will be larger than cost of the ON or OFF state. Also at time 0, the INT state will have a larger cost than ON state and since both cost curves are growing at a linear rate, the INT state cost will be larger than the ON state for the entire duration from 0 to 1, thus using the INT state in that duration will not yield the optimal cost. If $a + d = 1$, then similar to when $a + d > 1$, in the duration 0 to 1, the INT state will have a larger cost than the ON state and will have equal costs at time 1. Whether $a + d = 1$ or $a + d > 1$, after

time $t = 1$ the OFF state will be the state with the minimum cost. And thus, if the offline algorithm uses the INT state from 0 to 1, it will obtain a larger cost rather than using the ON state which leads to a contradiction. □

Thus when $a + d = 1$, we have $x_{opt_1} = x_{opt_2}$ and thus INT state is never used. The only situations where the offline algorithm uses the INT state is when $a + d < 1$, so now we can update the optimal threshold times to $x_{opt_1} = \min\{d/(1-a), 1\}$ and $x_{opt_2} = \max\{(1-d)/a, 1\}$. For the online algorithm we need to determine its x_1 and x_2 values that minimizes the competitive ratio. In the online model, the machine will start in the ON state will switch to INT and OFF state based on the values for x_1 and x_2. The $Cost_{online}$ will be

$$Cost_{online}(t) = \begin{cases} t & \text{if } t < x_1 \\ x_1 + a(t - x_1) + d & \text{if } x_1 \leq t < x_2 \\ x_1 + a(x_2 - x_1) + 1 & \text{if } t \geq x_2 \end{cases}$$

Lemma 4.0.3. *In order to have an optimal competitive ratio, $x_1 \leq x_{opt_1}$ must hold.*

Proof. For the case of $x_1 \leq x_{opt_1}$, we assume that $x_1 > x_{opt_1}$. If this is the case, $x_{opt_1} = x_1 + \delta$ such that $\delta > 0$. Then we have the following competitive ratio:

$$\frac{x_{opt_1} + \delta + d}{a(x_{opt_1} + \delta) + d}$$

The online and offline costs can be compared to get:

$$x_{opt_1} + \delta + d \geq a(x_{opt_1} + \delta) + d$$

$$x_{opt_1} + \delta \geq a(x_{opt_1} + \delta)$$

It is clear that as $\delta > 0$ increases the competitive ratio increases since the online costs increases at a faster rate than the offline cost. □

Lemma 4.0.4. *A necessary condition for the optimal competitive ratio is when $x_2 = x_{opt_2}$ holds.*

Proof. We assume the contrary that $x_2 < x_{opt_1}$ and therefore $x_2 = x_{opt_1} - \delta$ where $\delta > 0$, so the competitive ratio would be:

$$\frac{x_1 + a(x_{opt_1} - \delta - x_1) + 1}{x_{opt_1} - \delta}$$

$$a + \frac{x_1(1-a) + 1}{x_{opt_1} - \delta}$$

As δ increase the competitive ratio increases as well and $x_2 \geq x_{opt_1} \geq x_1$. So now to show that $x_2 = x_{opt_2}$ must be true, we first assume $x_2 > x_{opt_2}$, so the competitive ratio would be $x_1 + a(x_2 + \delta - x_1) + 1$.

It is clear that as δ increases, the competitive ratio will increase linearly. Now we assume the examine the competitive ratio if $x_{opt_1} \leq x_2 < x_{opt_2}$. So $x_2 = x_{opt_2} - \delta$. The competitive ratio would be:

$$\frac{x_1 + a(x_{opt_2} - \delta - x_1) + 1}{a(x_{opt_2} - \delta) + d}$$

$$1 + \frac{x_1(1-a) + 1 - d}{a(x_{opt_2} - \delta) + d}$$

Once again, as δ increases the competitive ratio is increasing. So when $x_2 > x_{opt_2}$ and $x_1 \leq x_2 < x_{opt_2}$ both lead to contradictions because the competitive ratio will not minimal in those cases thus $x_2 = x_{opt_2}$ to minimize the competitive ratio. □

From lemma 4.0.4, the competitive ratio for the 3 state machine depends only on the value of x_1, since it is known the competitive ratio is minimal when $x_2 = x_{opt_2}$ and x_{opt_2} can be computed by only knowing the values of a and d. Given an x_1 and x_2, the maximum cost for the online algorithm at time x_1 and x_2 will be $x_1 + d$ and $x_1 + a(x_2 - x_1) + 1$ respectively. The optimal cost offline cost can be computed at times x_1 and x_2 using x_{opt_1} and x_{opt_2}, so the competitive ratios can be derived for those two intervals, which will be denoted by CR_1 and CR_2.

$$CR_1 = \frac{x_1 + d}{x_1} \tag{4.1}$$

$$CR_2 = x_1 + a(x_2 - x_1) + 1 \tag{4.2}$$

The goal is to minimize the worst case competitive ratio for the two switching times. The competitive ratio of the system will be $\max\{CR_1, CR_2\}$.

Lemma 4.0.5. *The competitive ratio for the 3 state machine is minimized when $CR_1 = CR_2$.*

Proof. It will be assumed that $CR_1 \neq CR_2$ then either CR_1 or CR_2 will have the greater value. We will first assume that $CR_1 < CR_2$ and the competitive ratio will be minimal.

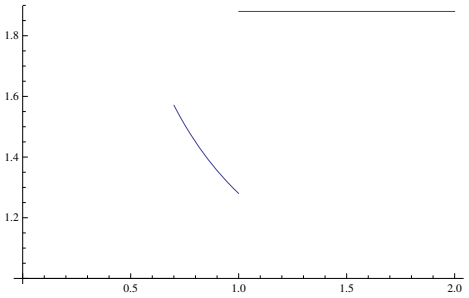

Figure 4.7: Competitive ratio $CR_1 < CR_2$

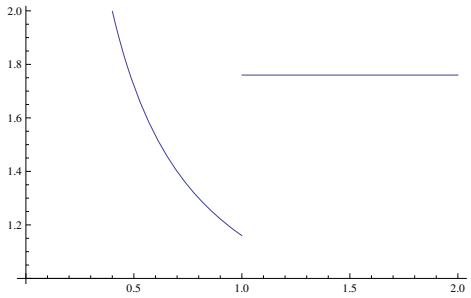

Figure 4.8: Competitive ratio $CR_1 > CR_2$

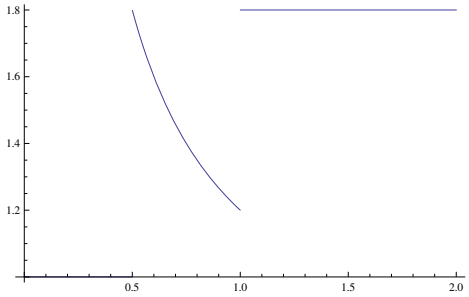

Figure 4.9: Competitive ratio $CR_1 = CR_2$

In Figure 4.7, if the value for x_1 would be decreased by an arbitrarily small constant such that $CR_1 < CR_2$ is still preserved, the value for CR_1 would increase but the value for CR_2 would decrease from (4.1) and (4.2) respectively. This leads to a contradiction because the competitive ratio was not minimal. In figure 4.8, the value of x_1 can be increased while still maintaining $CR_1 < CR_2$. This also leads to a contradiction since the competitive ratio between CR_1 and CR_2, the maximum of the two, has decreased. So when $CR_1 \neq CR_2$ the competitive ratio is not minimal and so it can only be minimal if $CR_1 = CR_2$. □

Lemma 4.0.6. *For an n state power down problem, the competitive ratio is minimized when* $CR_1 = CR_2 = ... = CR_n$.

Proof. Here we extend lemma 4.0.5, from 3 states to n states. Following lemma 4.0.5, we will assume the contrary that the competitive ratio is minimized when $CR_1 \neq CR_2 \neq ... \neq CR_n$ holds. We know that same principle applies with n states as it does with 3 states, that the competitive ratio for n states will be $\max_{i=1}^{n}\{CR_i\}$, so for some state j the competitive ratio CR_j will be the maximum of all the competitive ratios. In order to modify the competitive ratio, we choose a different x_j

value which causes the system to transition from state j to $j+1$ at a different time, we will choose a larger x_j value. Applying this will cause the competitive ratio CR_j to decrease and CR_{j+1} to increase while still maintaining CR_j as the maximum competitive ratio. Thus if we do a search for the maximum competitive ratio again with this new x_j value, $max_{i=1}^{n}\{CR_i\}$ CR_j will remain the largest competitive ratio but the overall competitive ratio for the system has been decreased so the initial assumption was incorrect because we obtained a smaller competitive ratio so $CR_1 = CR_2 = ... = CR_n$ must hold in order to have the minimal competitive ratio. □

Returning to the 3 state problem, the only way the cost of the online algorithm can be minimized if the value of x_1 is optimal. The values of a and d can be any value between 0 and 1, and $a+d=\lambda$. To have the optimal competitive ratio we know that $CR_1 = CR_2$ must be true. The value of x_2 is known, and setting $CR_1 = CR_2$ is used to obtain the optimal x_1 value.

$$\frac{x_1+d}{x_1} = x_1 + a(x_2 - x_1) + 1$$

And now we can solve for x_1. We can also rewrite the equation since $a = \lambda - d$.

$$x_1 = \frac{ax_2 - \sqrt{4d - 4ad + a^2x_2^2}}{2(a-1)} = \frac{(\lambda-d)x_2 - \sqrt{4d - 4d(\lambda-d) + x_2^2(\lambda-d)^2}}{2(\lambda-d-1)} \tag{4.3}$$

Using the value of x_1, we can substitute (4.3), into CR_1 or CR_2 and that will yield the optimal competitive ratio given a value for a and d.

$$CR_{opt} = 1 + \frac{2d(\lambda-d-1)}{(\lambda-d)x_2 - \sqrt{4d - 4d(\lambda-d) + x_2^2(\lambda-d)^2}} \tag{4.4}$$

Using equation (4.4), we can give a value for λ, and search for the optimal a and d values that will minimize the competitive ratios.

Theorem 4.0.7. *For a 3 state system where $a+d=1$, the optimal competitive ratio is achieved when $a = \frac{3}{5}$ and $d = \frac{2}{5}$.*

Proof. Consider the optimal competitive ratio in equation (4.4). Since $\lambda = 1$, we can compute the competitive ratios for all possible values of d.

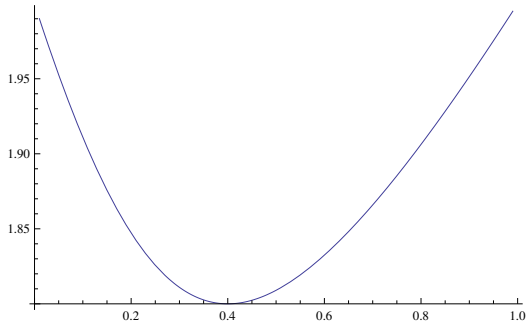

Figure 4.10: Competitive ratio for various d values

Thus it can be seen that when $d = 0.4$, it is optimized, we can compute the value for when the slope of the concave up parabola is 0, which will the value of d in which the competitive ratio is minimized.

$$CR_{opt}\Big(\frac{d}{dd}\Big) = \frac{10d - 2}{4\sqrt{5d^2 - 2d + 1}} - \frac{1}{2} = 0$$

After simplifying the above expression we get

$$20d^2 + 8d = 0$$

Solving for d, the value $d = \frac{2}{5}$ solves the above equation, and simultaneously we have $a = \frac{3}{5}$ since $a + d = 1$, and with those values the competitive ratio will be 1.8 which is the best known upper bound for a 3 state system. □

In Tables 4.2 and 4.3, we tabulate the minimum competitive ratio for various λ values where $\lambda = a + d$.

a	d	λ	CR	x_1	x_2
0.0512	0.0488	0.1	1.9976	0.0489	18.580
0.1046	0.0954	0.2	1.9908	0.0963	8.6487
0.1600	0.1400	0.3	1.9800	0.1429	5.3750
0.2173	0.1827	0.4	1.9654	0.1893	3.7613
0.2764	0.2236	0.5	1.9472	0.2361	2.8090
0.3373	0.2627	0.6	1.9254	0.2839	2.1859
0.4000	0.3000	0.7	1.9000	0.3333	1.7500
0.4646	0.3354	0.8	1.8708	0.3852	1.4305
0.5312	0.3688	0.9	1.8376	0.4403	1.1883
0.6000	0.4000	1.0	1.8000	0.5000	1.000
0.6312	0.4688	1.1	1.8376	0.5597	1.0000
0.6646	0.5354	1.2	1.8708	0.6148	1.0000
0.7000	0.6000	1.3	1.9000	0.6667	1.0000
0.7373	0.6627	1.4	1.9254	0.7161	1.0000
0.7764	0.7236	1.5	1.9472	0.7639	1.0000
0.8172	0.7827	1.6	1.9654	0.8107	1.0000
0.8600	0.8400	1.7	1.9800	0.8571	1.0000
0.9046	0.8954	1.8	1.9908	0.9037	1.0000

Table 4.2: Experimental results for a given λ value

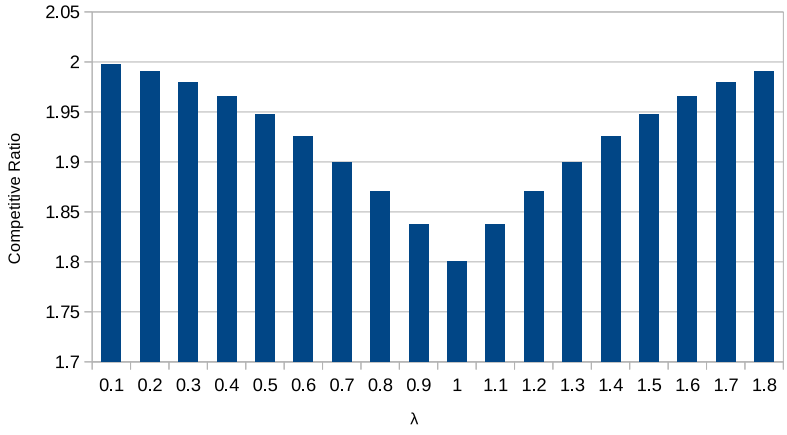

Figure 4.11: Optimal competitive ratios for given λ values

a	d	λ	Competitive ratio	x_1	x_2
0.565305	0.384695	0.95	1.81939	0.46949	1.088447829
0.572196	0.387804	0.96	1.81561	0.475479	1.0699061161
0.579111	0.390889	0.97	1.81178	0.481522	1.0518035403
0.58605	0.39395	0.98	1.8079	0.487622	1.034126781
0.593012	0.396988	0.99	1.80398	0.493781	1.0168630652
0.6	0.4	1	1.8	0.5	1
0.603012	0.406988	1.01	1.80398	0.50622	1
0.60605	0.41395	1.02	1.8079	0.512377	1
0.609111	0.420889	1.03	1.81178	0.518478	1
0.612196	0.427804	1.04	1.81561	0.524522	1
0.615305	0.434695	1.05	1.81939	0.530511	1

Table 4.3: Experimental results for λ values close to 1

In tables 4.2 and 4.3 and figure 4.11, if the optimal competitive ratios are found for several λ values, the overall optimal competitive ratio is obtained when $\lambda = 1$. Based on those experimental results, we will assume for a 3 state machine, the competitive ratio is optimal when $a + d = 1$.

Chapter 5

The Five State Problem

5.1 Preliminaries

In this chapter, we take the basic concepts from the three system system and we apply them onto a five state system. When we consider the five state problem we have five power states where we have an on state s_0, an off state s_4, and a set of intermediate states s_1, s_2, and s_3 where the higher index denotes a lower power state. Each state will have a set of unit costs and power up costs and $a_0 > a_1 \geq a_2 \geq a_3 > a_4$ where $a_4 = 0$ since it is the off state and we will normalize the costs once again so $a_0 = 1$. For the power up costs the following will be true $d_0 < d_1 \leq d_2 \leq d_3 < d_4$ where $d_0 = 0$ since it is the cost for the on state to power up to the on state and $d_4 = 1$ since the highest power up cost is from the off state to the on state.

We have a set of transition times, x_1, x_2, x_3, and x_4 where each x_i denotes a transition from s_{i-1} to s_i. We will have a set of times x_{opt_1}, x_{opt_2}, x_{opt_3}, and x_{opt_4} which will be used for the optimal offline algorithm to determine the optimal cost based on the wait time. Given the value for t, the idle time duration, the online cost will be computed in the following way:

$$\mathcal{A}(\mathrm{t}) = \begin{cases} a_0 t & \text{if } t < x_1 \\ a_0 x_1 + a_1(t - x_1) + d_1 & \text{if } x_1 \leq t < x_2 \\ a_0 x_1 + a_1(x_2 - x_1) + a_2(t - x_2) + d_2 & \text{if } x_2 \leq t < x_3 \\ a_0 x_1 + a_1(x_2 - x_1) + a_2(x_3 - x_2) + a_3(t - x_3) + d_3 & \text{if } x_3 \leq t < x_4 \\ a_0 x_1 + a_1(x_2 - x_1) + a_2(x_3 - x_2) + a_3(x_4 - x_3) + d_4 & \text{if } t \geq x_4 \end{cases}$$

The offline cost will be $\mathcal{OPT}(\mathrm{t}) = \min\{a_0 t, a_1 t + d_1, a_2 t + d_2, a_3 t + d_3, d_4\}$. As we know from lemma 3.0.1 the competitive ratio is maximized at transition time and from lemma 4.0.6, the competitive

ratio can only be minimized if the competitive ratio for each transition time x_i are equal. The competitive ratio for each transition time will be denoted by CR, as seen in the following:

$$\frac{x_1 + d_1}{x_1} = CR \tag{5.1}$$

$$\frac{x_1 + a_1(x_2 - x_1) + d_2}{\min\{x_2, a_1 x_2 + d_1\}} = CR \tag{5.2}$$

$$\frac{x_1 + a_1(x_2 - x_1) + a_2(x_3 - x_2) + d_3}{\min\{x_3, a_1 x_3 + d_1, a_2 x_3 + d_2\}} = CR \tag{5.3}$$

$$\frac{x_1 + a_1(x_2 - x_1) + a_2(x_3 - x_2) + a_3(x_4 - x_3) + d_4}{d_4} = CR \tag{5.4}$$

We solve for x_1, x_2, x_3, and x_4 in equations 5.1, 5.2, 5.3, and 5.4 respectively. Let $\alpha = x_1(1 - a_1)$, $\beta = x_1(1 - a_1) + x_2(a_1 - a_2)$, $\beta' = x_1(a_1 - 1) + x_2(a_2 - a_1)$, $\gamma = \beta' + x_3(a_3 - a_2)$. We obtain:

$$x_1 = \frac{d1}{CR - 1} \tag{5.5}$$

$$x_2 = \max\left\{\frac{\alpha + d_2}{CR - a_1}, \frac{\alpha + d_2 - CR \cdot d_1}{a_1(CR - 1)}\right\} \tag{5.6}$$

$$x_3 = \max\left\{\frac{\beta + d_3}{CR - a_2}, \frac{\beta' - d_3 + CR \cdot d_1}{a_2 - a_1 \cdot CR}, \frac{\beta + d_3 - CR \cdot d_2}{a_2(CR - 1)}\right\} \tag{5.7}$$

$$x_4 = \frac{\gamma + CR - 1}{a_3} \tag{5.8}$$

Notice that in equations 5.2 and 5.3 we have several several expressions for the optimal cost and in equations 5.6 and 5.7 we have several possible values for x_2 and x_3. Lemmas 9.3.1 and 9.3.2 proved that when the minimal optimal cost was used to compute the online standby time, that standby time would yield the maximum of all possible standby times. And thus, we use the maximum standby time for x_2 and x_3. As for x_4, we do not consider all the possible offline costs, in lemma 4.0.4 for the three state problem, it was proven that $x_2 = x_{opt_2}$ must hold and is a necessary condition to have an optimal competitive ratio, similar holds for the five state problem, we can only obtain the optimal offline cost when $x_4 = x_{opt_4}$, for the same reason as with the three state problem, which was shown in lemma 4.0.4. Thus, we do not consider x_4 in an event if one of the following x_4, $a_1 x_4 + d_1$, $a_2 x_4 + d_2$, and $a_3 x_4 + d_3$ is the minimal cost because that would imply that $x_4 < x_{opt_4}$ which we know will not result in the optimal competitive ratio. The optimal times can be computed in the

following way:

$$x_{opt_4} : \max_t : d_4 = \begin{cases} a_0 t \\ a_1 t + d_1 \\ a_2 t + d_2 \\ a_3 t + d_3 \end{cases} \tag{5.9}$$

These will determine the state of the offline algorithm based on the wait time of the net request. So if the next request occurs between x_{opt_i} and $x_{opt_{i+1}}$ then the offline algorithm will be in state i until the request arrives.

5.2 The Five State Power Down Algorithm

For any state machine, there exists a schedule that is $(3 + 2\sqrt{2})$ - competitive. [9] Our goal is to find the minimal competitive ratio in the range $[1, 3 + 2\sqrt{2}]$. Given the 5 states and their respective a_i and d_i values, we assign a value for CR and we compute the standby times using equations 5.5, 5.6, 5.7, and 5.8. The idea of the algorithm is we use some value for the competitive ratio, and then we compute the standby times and we obtain switch times that produce the given competitive ratio. This however may not be the optimal competitive ratio for this system. If the value of $x_4 > x_{opt_4} + \theta$, where $\theta > 0$ is an arbitrarily small constant, then we can choose a new CR that is smaller than the value we assigned earlier since we know that the competitive ratio can only be minimal if $x_4 = x_{opt_4}$. However, if $x_4 < x_{opt_4}$ then there does not exist a schedule for this five state system that will be CR-competitive so we need to choose a new competitive ratio that is larger than CR. We keep applying the strategy until we find a schedule such that $x_{opt_4} \leq x_4 \leq x_{opt_4} + \theta$. The algorithm essentially will be an approximation algorithm so we may not obtain the exact minimal competitive ratio but rather a competitive ratio that will be arbitrarily close the optimal competitive ratio. Here is the sketch of the five state power down problem.

Data: Given values $a_{0\to4}$, and $d_{0\to4}$

lowerBound = 1, upperBound = $3 + 2\sqrt{2}$;

CR = (leftEnd + rightEnd) / 2;

Compute x_1, x_2, x_3, and x_4 using CR;

while $x_4 < x_{opt_4}$ *or* $x_4 > x_{opt_4} + \theta$ **do**

  if $x_4 < x_{opt_4}$ **then**

    lowerBound = CR;

  else

    upperBound = CR;

  end

  CR = (lowerBound + upperBound) / 2;

  Recalculate x_1, x_2, x_3, and x_4 using the updated CR value;

end

Algorithm 1: Five State Power Down Algorithm

We will execute algorithm 1, for a five state system with the given standby and power up costs, $a_0 = 1$ $d_0 = 0$ $a_1 = 0.55$ $d_1 = 0.225$ $a_2 = 0.4$ $d_2 = 0.4$ $a_3 = 0.25$ $d_3 = 0.60$ $a_4 = 0$ $d_4 = 1$. The value of $x_{opt_4} = 1.6$ and $\theta = 0.01$. Each iteration of algorithm 1 is tabulated below.

Iteration	x_1	x_2	x_3	x_4	lowerBound	upperBound	CR
1	0.0932	0.1543	0.2207	9.2632	1.000	5.828	3.414
2	0.1864	0.2920	0.4027	4.0756	1.000	3.414	2.207
3	0.3731	0.6249	1.0401	0.7414	1.000	2.207	1.603
4	0.2486	0.3778	0.5248	2.6309	1.603	2.207	1.905
5	0.2984	0.4438	0.7193	1.7810	1.603	1.905	1.754
6	0.3314	0.4864	0.8488	1.3184	1.603	1.754	1.679
7	0.3142	0.4643	0.7815	1.5509	1.679	1.754	1.716
8	0.3061	0.4538	0.7496	1.6670	1.716	1.754	1.735
9	0.3099	0.4587	0.7645	1.6122	1.716	1.735	1.726
10	0.3121	0.4615	0.7729	1.5816	1.716	1.726	1.721
11	0.3108	0.4598	0.7678	1.6000	1.721	1.726	1.724

Table 5.1: Execution of algorithm 1 with sample input

When we look at table 5.1, we are essentially performing a binary search at each iteration. At the

last iteration we obtain a value for x_4 that is within the $x_{opt_4} + \theta$ or rather in this case $x_4 = x_{opt_4}$. We notice that when $x_4 < x_{opt_4}$ the competitive ratio is smaller than when $x_4 \geq x_{opt_4}$, which is a logical error. This occurs only because when we compute x_4 from equation 5.8, we only considered the offline cost of d_4, we did not consider costs x_4, $a_1x_4 + d_1$, $a_2x_4 + d_2$, or $a_3x_4 + d_3$. Which means the true optimal cost, a cost less than d_4, was not used to compute x_4 which results in a lower competitive ratio than the actual optimal competitive ratio.

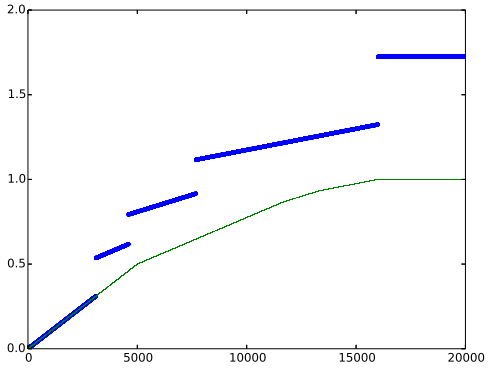

Figure 5.1: Online and offline costs for example 5 state system

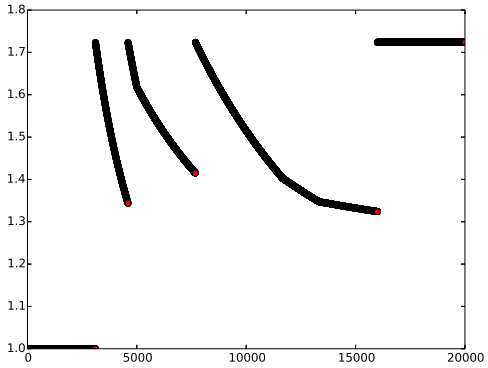

Figure 5.2: Competitive Ratio For Various Standby Times

Figure 5.1 shows the cost of the online and offline algorithms for the five state machine and figure 5.2 displays the competitive ratio for using the optimal transition times we obtained from table 5.1. We can see that the competitive ratio is always maximized at the transition time, and that the competitive ratio decreases when the standby time diverges away from any transition time. The competitive ratio will continue to decrease until the standby reaches the next transition time. Now we will show the optimal transition times for various sets of idle and power up costs in the following tables.

i	a	d	x	CR	a	d	x	CR
0	1.0000	0.0000	0.0000		0.0000	1.0000	0.0000	
1	0.7500	0.2500	0.3566		0.6000	0.2000	0.2706	
2	0.5000	0.5000	0.6195	1.701	0.4000	0.4000	0.4462	1.739
3	0.2500	0.7500	0.8277		0.2000	0.6000	0.6990	
4	0.0000	1.0000	1.0001		0.0000	1.0000	2.0086	
i	a	d	x	CR	a	d	x	CR
0	1.0000	0.0000	0.0000		1.0000	0.0000	0.0000	
1	0.6000	0.1000	0.1290		0.7000	0.2000	0.2614	
2	0.4000	0.3000	0.3744	1.775	0.3000	0.4000	0.4492	1.765
3	0.1000	0.6000	0.8256		0.1000	0.8000	1.5343	
4	0.0000	1.0000	4.0083		0.0000	1.0000	2.0001	
i	a	d	x	CR	a	d	x	CR
0	1.0000	0.0000	0.0000		1.0000	0.0000	0.0000	
1	0.8000	0.1000	0.1376		0.5500	0.2250	0.3108	
2	0.5000	0.4000	0.4614	1.7265	0.4000	0.4000	0.4598	1.724
3	0.1000	0.8000	0.9003		0.2500	0.6000	0.7678	
4	0.0000	1.0000	2.0043		0.0000	1.0000	1.6000	

Table 5.2: Optimal competitive ratio within $\theta = 0.01$ for various a and d costs

5.3 Comparing three state machine to five state machine

In this section we compare the 5 state machine, with various power state costs, with the best known 1.8-competitive three state machine. Let us compare the five state machine from table 5.2 where the competitive ratio is 1.701. The competitive ratios and power costs using the power states and a range of idle times is shown in the following graphs.

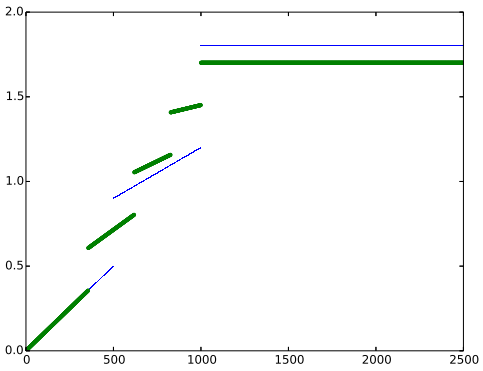

Figure 5.3: Costs for three state and five state machines for CR = 1.8 and CR = 1.701

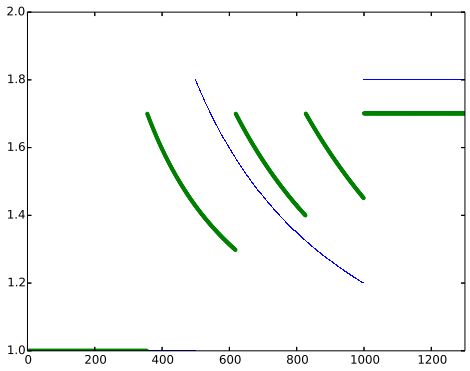

Figure 5.4: Competitive ratio for three state and five state machine for CR = 1.8 and CR = 1.701

From figures 5.3 and 5.4, we can see the the three state machine has better costs and competitive ratio except once the three state machine powers down to the off state, although there is a small duration when it transitions to the intermediate state, after its x_1 time, where it has a larger cost but once the five state machine switches to state 2 (the five state machine at time x_2), the five state machine has a larger cost but ultimately the three state machine is worse than the five state machine once it powers down. Once again from table 5.2, let us consider the five state machine has parameters such that its competitive ratio is 1.739, the following graphs show the behavior.

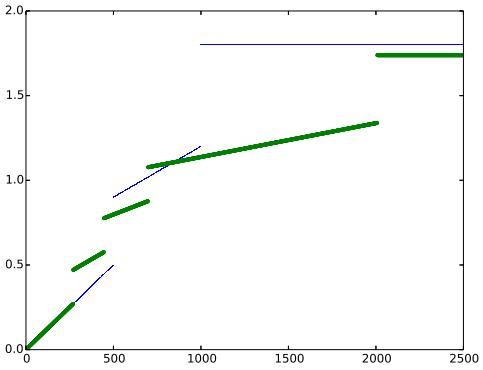

Figure 5.5: Costs for three state and five state machines for CR = 1.8 and CR = 1.739

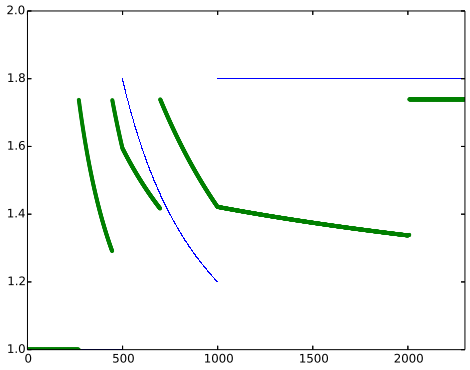

Figure 5.6: Competitive ratio for three state and five state machine for CR = 1.8 and CR = 1.739

In figures 5.5 and 5.6, we see more situations when the five state machine has better costs and better competitive ratios. Once again, the competitive ratio is worse for the three state machine for when it transitions to the off state. However in this case, we have a bigger difference, compared to the last example, in cost and competitive ratio at time 1, when the three state machine powers down. Also in this example, there are more situations where the five state machine has better cost than the three state machine. Now if we choose the the five state machine with competitive ratio 1.775, we have the following costs.

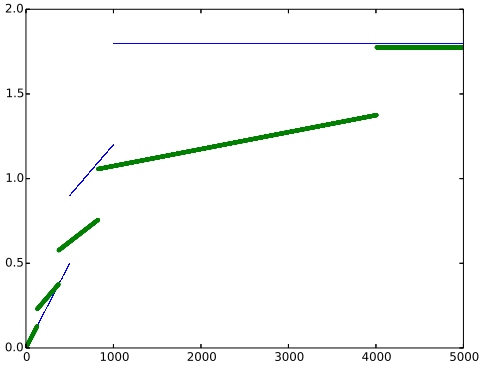

Figure 5.7: Costs for three state and five state machines for CR = 1.8 and CR = 1.775

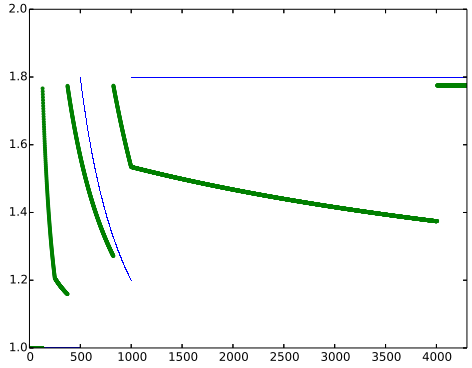

Figure 5.8: Competitive ratio for three state and five state machine for CR = 1.8 and CR = 1.775

We are seeing the same pattern emerge once again, the five state algorithm has better cost when the three state machine powers down to the off state and when the five state algorithm is in state 3, in between x_3 and x_4, the five state machine has lower costs. There are some moments when the five state machine has better cost than the three state at earlier times, but the greatest savings is in the duration x_3 to x_4. The next schedule from 5.2, when the five state machine that has competitive ratio 1.765, we have the following:

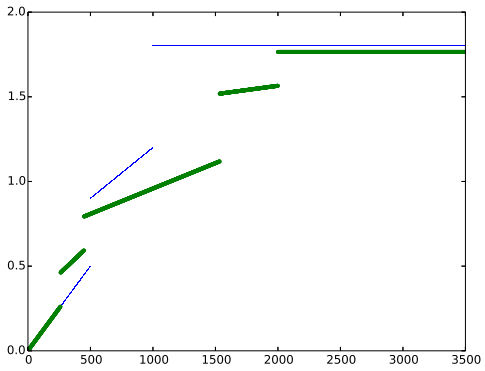

Figure 5.9: Costs for three state and five state machines for CR = 1.8 and CR = 1.765

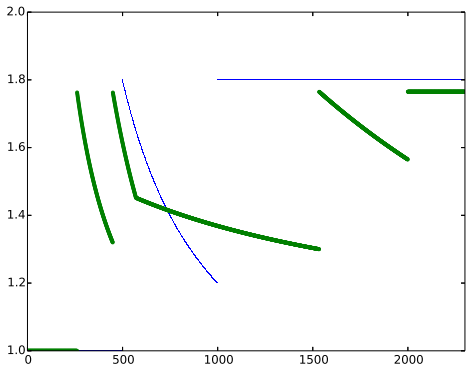

Figure 5.10: Competitive ratio for three state and five state machine for CR = 1.8 and CR = 1.765

In figures 5.9 and 5.10, we see that there are more savings intervals than in previous examples. From time x_2 and beyond (x_2 that was calculated in the five state system), the five state machine has a better cost than the three state machine. Although the savings seems to be greater in the duration when the three state machine is in the off state and the five state machine is in state 3 (the state right before off state). The next example is when the five state machine has competitive ratio 1.7265 from table 5.2, and we have the following figures:

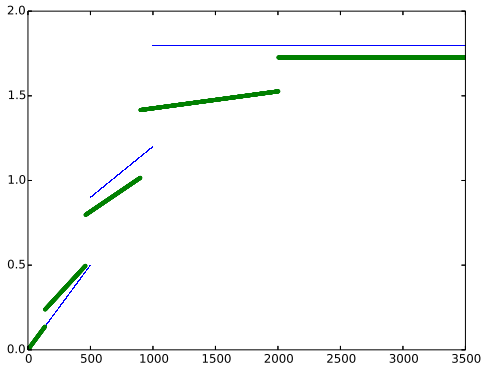

Figure 5.11: Costs for three state and five state machines for CR = 1.8 and CR = 1.7265

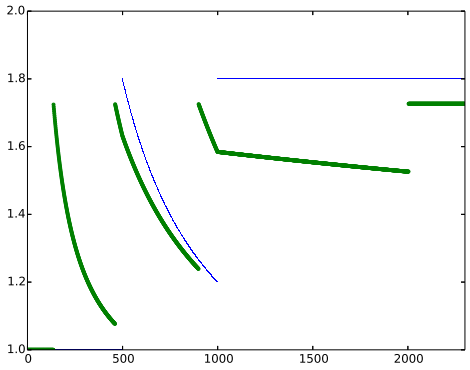

Figure 5.12: Competitive ratio for three state and five state machine for CR = 1.8 and CR = 1.7265

In this example, we see a similar pattern as with the last example. Now we will see how the costs and competitive ratios look between the three state and five state machine with competitive ratio 1.724 from table 5.2.

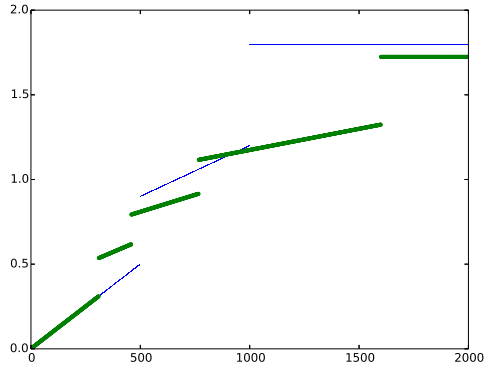

Figure 5.13: Costs for three state and five state machines for CR = 1.8 and CR = 1.724

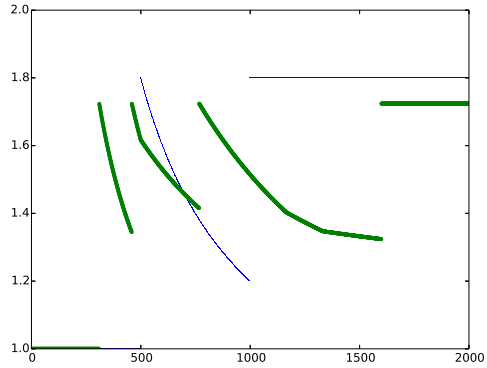

Figure 5.14: Competitive ratio for three state and five state machine for CR = 1.8 and CR = 1.724

Based on the best known three state cost algorithm and several five state algorithms used in the section, we can see that introducing extra states can be beneficial. Also we see that the five state algorithm has a favorable cost and competitive ratio once the three state machine transitions to the off state and usually the five state machine is in some higher power state other than off state. If the request arrives after the three state machine powers down to the off state and the request arrives between x_3 and x_4 (or in some cases x_2 and x_4), the five state machine will save power, and the difference of competitive ratio is increasing, since the competitive ratio of the five state system is decreasing in that interval as seen in Figure 5.14.

5.4 Comparing three state machine to five state machine where we increase its competitive ratio

In this section we will analyze the cost and competitive ratio of the five state machines with the three state machines, with increased competitive ratios of the five state machines. In the previous examples, we used five state machines with 1.701, 1.739, 1.775, 1.765, 1.7265, and 1.724 from table 5.2, here we will increase the competitive ratio to 1.8 in each system which will adjust the values for x_1, x_2, x_3, and x_4. In this scenario, both the three state machine and the five state machines that

is used in this analysis are both be 1.8-competitive. We first adjust the five state machine that is 1.701-competitive to 1.8-competitive, the following figures will show the costs and the competitive ratios.

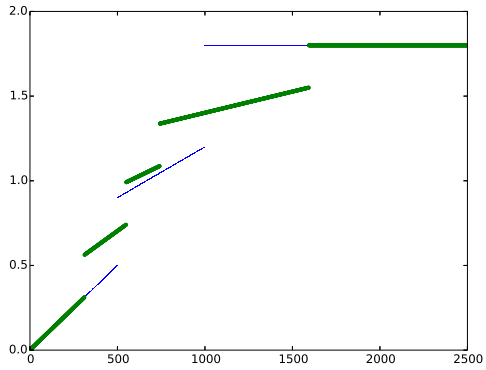

Figure 5.15: Costs for three state and five state machines for CR = 1.8 and CR = 1.701 raised to 1.8

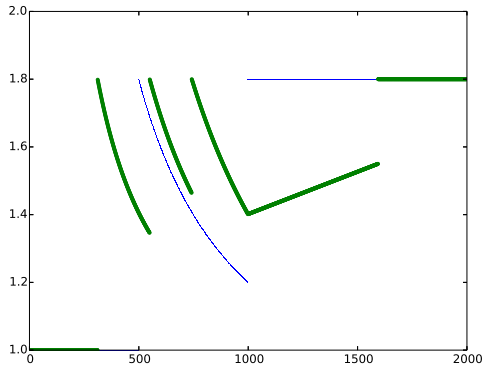

Figure 5.16: Competitive ratio for three state and five state machine for CR = 1.8 and CR = 1.701 raised to 1.8

We can see that the five state machine with the same competitive ratio has lower costs than the three state machine when it is in the off state. Here once they are both in the off state, they will have the same cost since they have the same competitive ratio. We see that the five state competitive ratio has a moment where it is increasing right before it transitions to the off state. This is due to the fact that the five state system is not optimal when we set the competitive ratio to 1.8 when calculating the transition times. However we can see some savings from x_3 to x_4 when we raise the competitive ratio to 1.8. Now we will increase the five state machine from table 5.2 which has competitive ratio 1.739 and we will also increase the competitive ratio to 1.8.

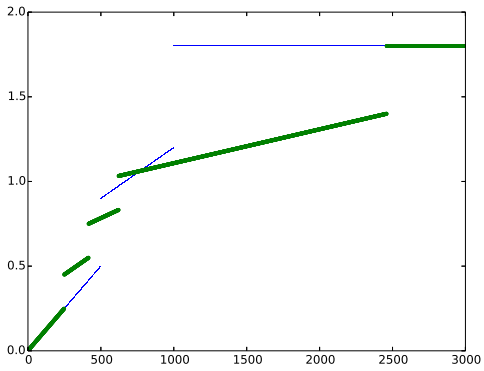

Figure 5.17: Costs for three state and five state machines for CR = 1.8 and CR = 1.739 raised to 1.8

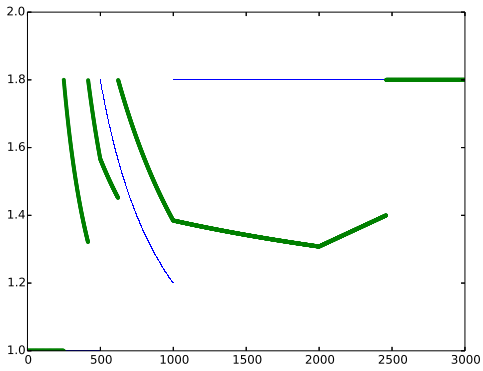

Figure 5.18: Competitive ratio for three state and five state machine for CR = 1.8 and CR = 1.739 raised to 1.8

From figures 5.17 and 5.18, we see similar behavior when the five state machine was 1.739 competitive. The five state machine with the same competitive ratio had better performance when the three state machine powered down. Let us take a look at the rest of the examples when the five state machines with competitive ratio 1.775, 1.765, 1.7265, and 1.724 from table 5.2 is raised up tp 1.8-competitive.

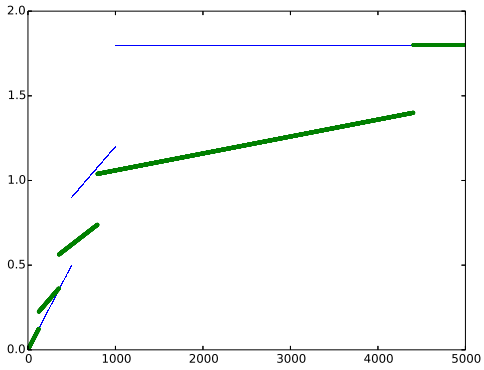

Figure 5.19: Costs for three state and five state machines for CR = 1.8 and CR = 1.775 raised to 1.8

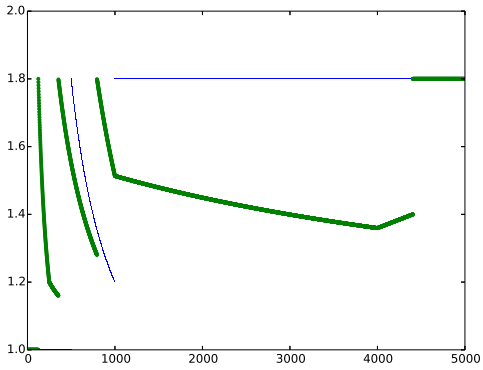

Figure 5.20: Competitive ratio for three state and five state machine for CR = 1.8 and CR = 1.775 raised to 1.8

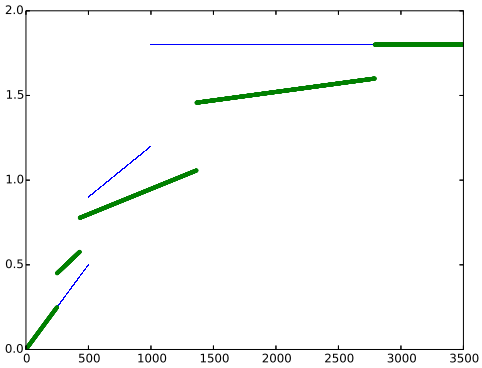

Figure 5.21: Costs for three state and five state machines for CR = 1.8 and CR = 1.765 raised to 1.8

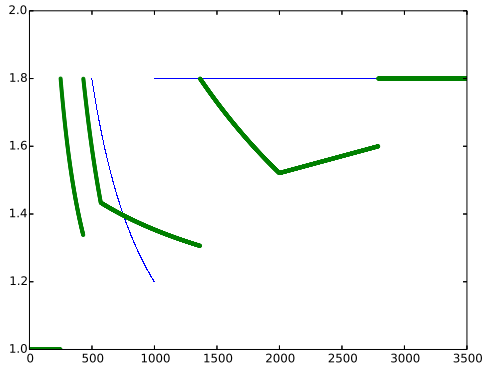

Figure 5.22: Competitive ratio for three state and five state machine for CR = 1.8 and CR = 1.765 raised to 1.8

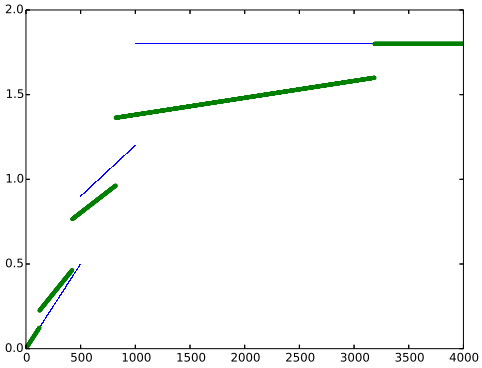

Figure 5.23: Costs for three state and five state machines for CR = 1.8 and CR = 1.7265 raised to 1.8

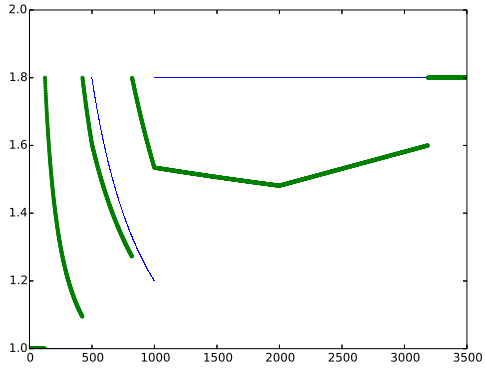

Figure 5.24: Competitive ratio for three state and five state machine for CR = 1.8 and CR = 1.7265 raised to 1.8

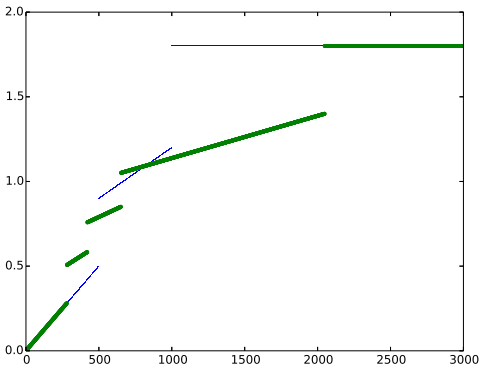

Figure 5.25: Costs for three state and five state machines for CR = 1.8 and CR = 1.724 raised to 1.8

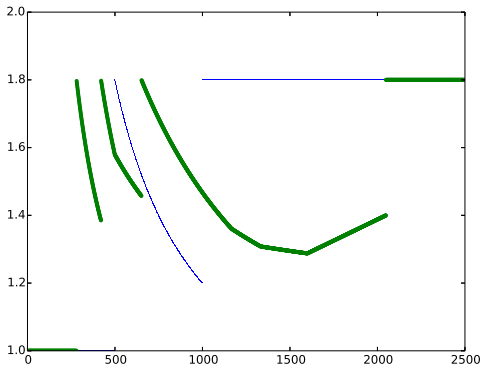

Figure 5.26: Competitive ratio for three state and five state machine for CR = 1.8 and CR = 1.724 raised to 1.8

In conclusion, considering our simulations, we can say that we get the same pattern that if the request arrives after the three state machine powers down and before the five state machine powers down, the five state machine has better results, its competitive ratio is smaller. However, before the three state machine powers down, whether or not we raise the five state machine to 1.8-competitive, the two systems have similar costs and competitive ratio where the three state machine seems to have the slight advantage. However when we do not raise the competitive ratio to 1.8 for the five state systems, we seem to have a favorable competitive ratio for more slack systems, when requests arrive after x_2 (the x_2 used in the three state machine), and the five state machine has no real advantage for busier systems, when the requests arrive before x_2 (the x_2 used in the three state machine).

Chapter 6

Continuous State Problem

In the continuous state problem, we choose transition times from a higher power state to a lower power state according to a continuous function for the idle costs and for the power up costs, in our analysis we choose an idle cost curve $a(r) = 1 - r^a$ and a power up cost curve $d(r) = cr^d$ where $r \in [0, 1]$ is the power state and a, d, $c > 0$ are control parameters. Here we use assign $a = 3$, $d = 5$, $c = 1.5$, and r is the power state being used. Figure 6.1 shows the two curves.

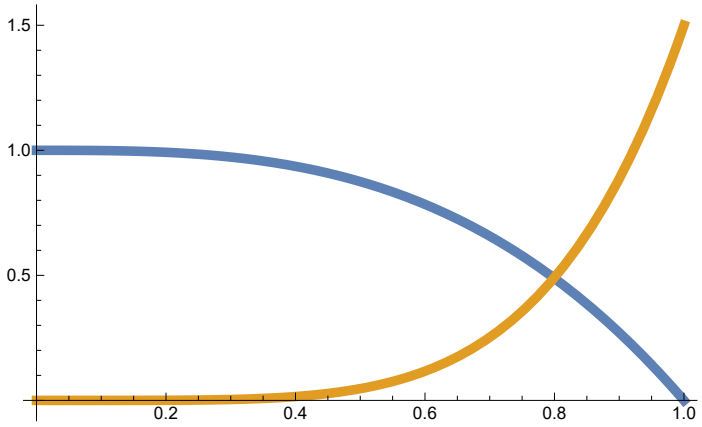

Figure 6.1: $a(r)$ and $d(r)$ curves

Using the two curves, we will develop a strategy for the online and offline model as to when to switch to lower power states while idling. As with the discrete model, the power state of the offline algorithm can chosen as soon as we have the idle period since the value of r will be known in advance, since we choose the state at the beginning of the idle period, we can simply take the derivative of $a(r) + d(r)$ and obtain the function $\text{Stategy}_{\text{OFF}}(r) = (\frac{a \cdot r}{d \cdot c})^{\frac{1}{d-a}}$ which will determine

the state for the offline algorithm to be optimal. As with the discrete model, we computed the time when both the online and offline algorithm is in the OFF state, for the continuous model the time when the machine powers down is $x_m = \frac{c \cdot d}{a}$, we derive this result from

$$\left(\frac{a \cdot x_m}{d \cdot c}\right)^{\frac{1}{d-a}} = 1 \tag{6.1}$$

$$\frac{a x_m}{C \cdot d} = 1 \tag{6.2}$$

$$x_m = \frac{c \cdot d}{a} \tag{6.3}$$

As with the discrete model, the online algorithm will start from its initial state and will transition to lower power states until the next request arrives, in our first experiment, we will have $\text{Strategy}_{\text{ONLINE}}(r) = \text{Strategy}_{\text{OFF}}(r)$, which will be a continuous version of the lower envelope algorithm. The cost of the offline algorithm will be $\text{Cost}_{\text{OFF}}(r) = r \cdot a(\text{Strategy}_{\text{OFF}}(r)) + d(\text{Strategy}_{\text{OFF}}(r))$ and the cost of the online algorithm will be $\text{Cost}_{\text{ONLINE}}(r) = \int_0^r a(\text{Strategy}_{\text{ONLINE}}(r))dr + d(\text{Strategy}_{\text{ONLINE}}(r))$.

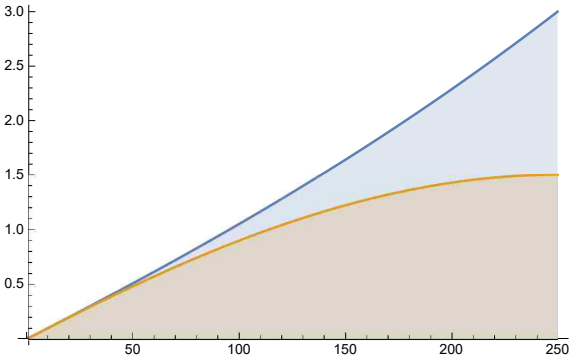

Figure 6.2: Cost of OPT and ONLINE

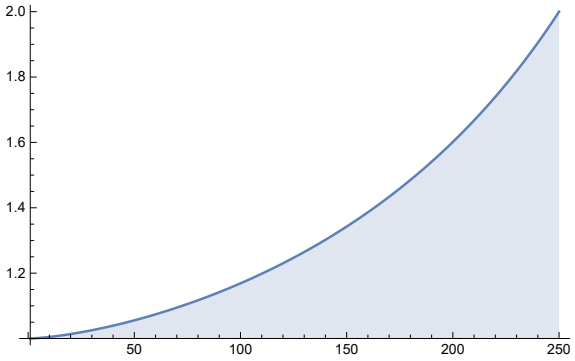

Figure 6.3: Competitive ratio

We see that the competitive ratio shown in Figure 6.3 shows the competitive ratio growing as the idle time increases and at time $x_m = 2.5$ the competitive ratio is 2, which is consistent with the discrete lower envelope algorithm. As we saw with the 3 state problem, we can adjust the switch times to earlier values than the time chosen by the lower envelope algorithm to obtain a better competitive ratio, we choose the following strategy for the online algorithm

$$\text{Strategy}_{t,z}(r) = \text{Stategy}_{\text{OFF}}(r) + \text{Strategy}_{\text{OFFLINE}}(r)^{z+t} - \text{Strategy}_{\text{OFFLINE}}(r)^{1+z} \tag{6.4}$$

where the values of t and z determine the rate at which the online strategy changes its power states. When we decrease the value of t and z, the online algorithm transitions to a lower power

state rapidly, but when we decrease the values of t and z, the online strategy transitions to a lower power state less rapidly. The value of t intensives the behavior more than the value of z. We show a few examples.

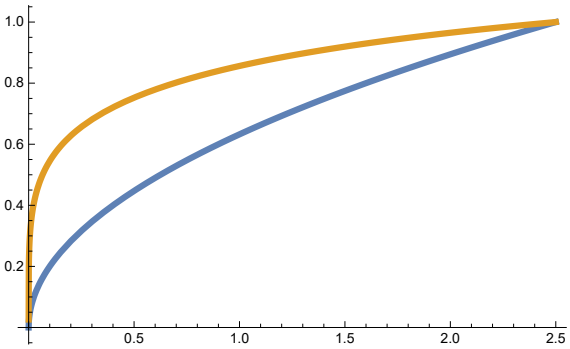

Figure 6.4: OPT and Online strategies for $t = 0.312$, $z = 0.1$

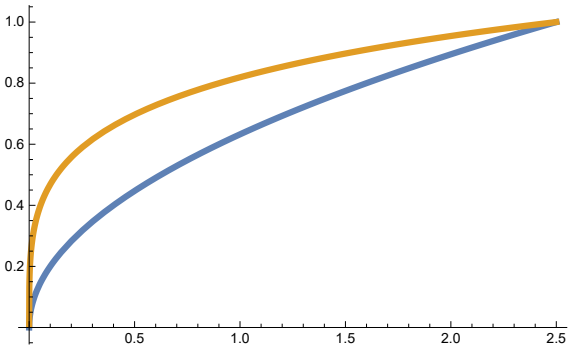

Figure 6.5: OPT and Online strategies for $t = 0.412$, $z = 0.1$

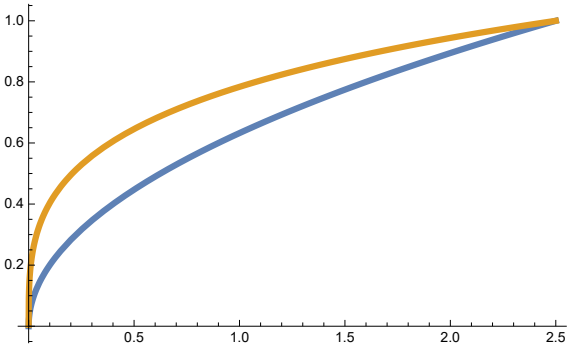

Figure 6.6: OPT and Online strategies for $t = 0.512$, $z = 0.1$

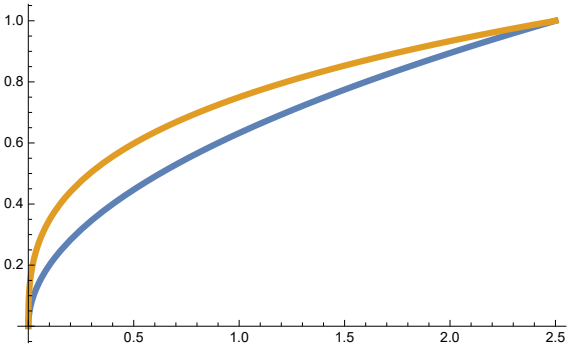

Figure 6.7: OPT and Online strategies for $t = 0.612$, $z = 0.1$

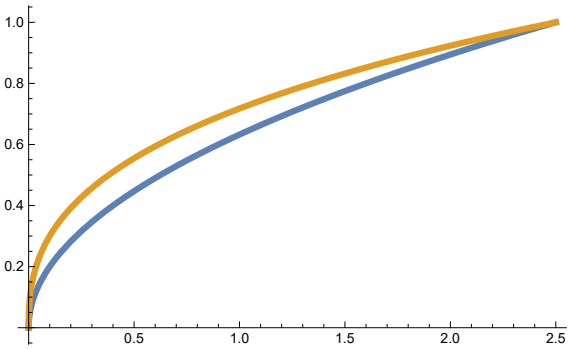

Figure 6.8: OPT and Online strategies for $t = 0.712$, $z = 0.1$

In figures 6.4, 6.5, 6.6, 6.7, and 6.8 we see the differences between the offline and online schedules. As we increase the value of t, we see that the online strategy decreases the rate at which it transitions to lower power states. Let us see the costs for the strategies.

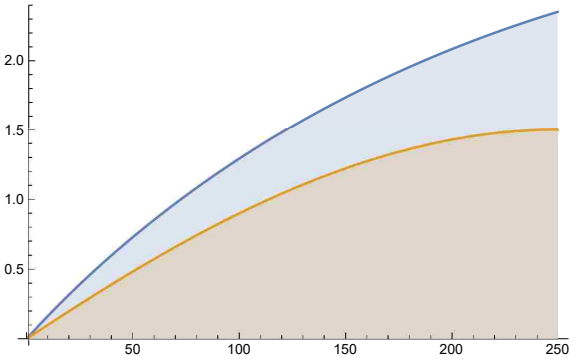

Figure 6.9: Cost of OPT and Online for $t = 0.312$, $z = 0.1$

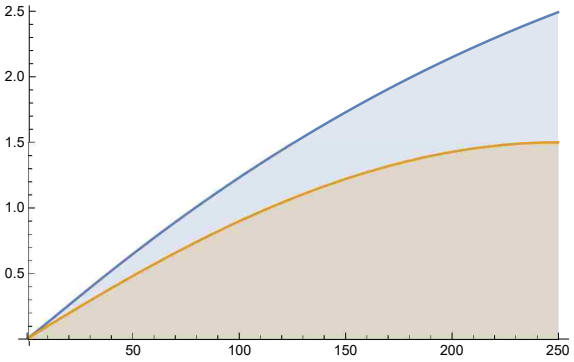

Figure 6.10: Cost of OPT and Online for $t = 0.412$, $z = 0.1$

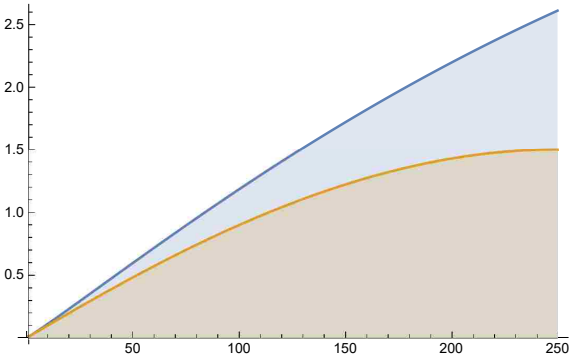

Figure 6.11: Cost of OPT and Online for $t = 0.512$, $z = 0.1$

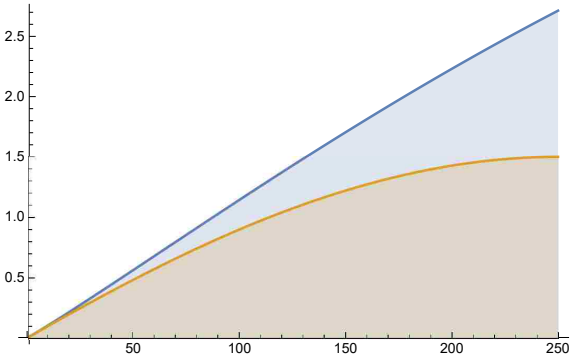

Figure 6.12: Cost of OPT and Online for $t = 0.612$, $z = 0.1$

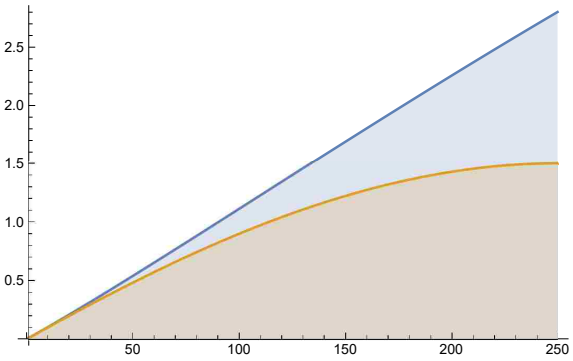

Figure 6.13: Cost of OPT and Online for $t = 0.712$, $z = 0.1$

We see that when t is larger, the costs at the beginning of the idle duration of both the online and offline algorithm are similar. As the idle duration increases, the cost for a higher t value increases the cost of the online algorithm. Let us see the competitive ratio of the strategies.

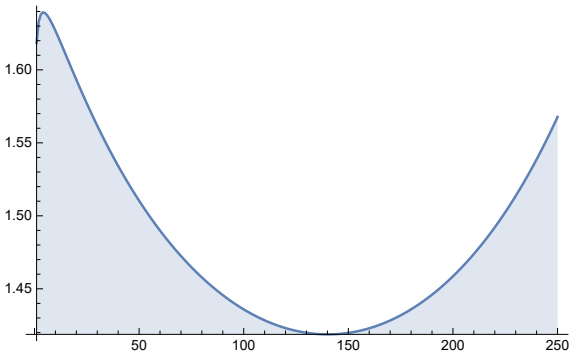

Figure 6.14: Competitive ratio $t = 0.312$, $z = 0.1$

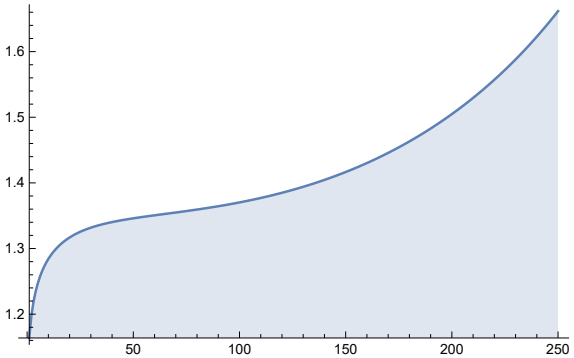

Figure 6.15: Competitive ratio $t = 0.412$, $z = 0.1$

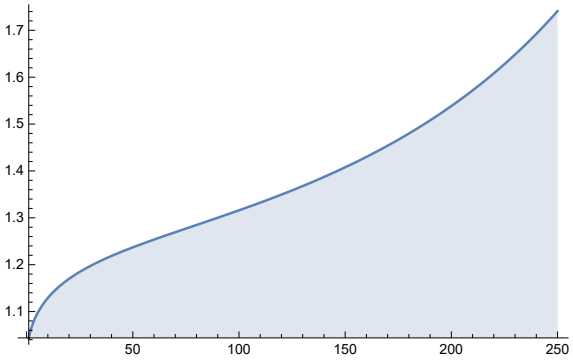

Figure 6.16: Competitive ratio $t = 0.512$, $z = 0.1$

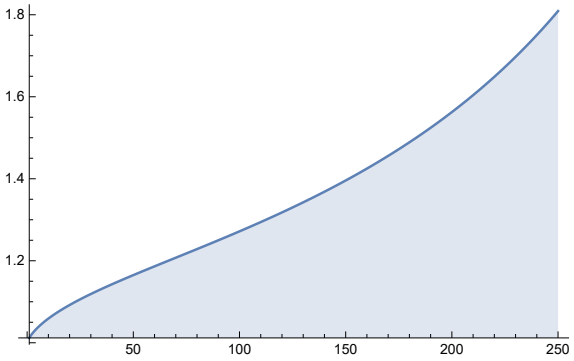

Figure 6.17: Competitive ratio $t = 0.612$, $z = 0.1$

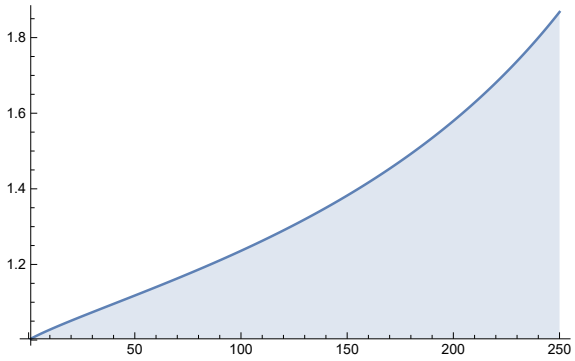

Figure 6.18: Competitive ratio $t = 0.712$, $z = 0.1$

The competitive ratio curves are consistent with the cost curves the cost and the competitive ratio of the online algorithms using the strategy with smaller t values has a smaller competitive ratio. When t is larger, then the rate at which the online algorithm changes states is similar to that of the offline algorithm strategy, and hence the behavior is similar to the LEA, where the competitive ratio initially has a smaller value at the beginning but then increases as the idle duration increases, however if the online strategy transitions at a faster rate than the offline algorithm, we obtain favorable results for the competitive ratio. Now we adjust the z values and observe the results.

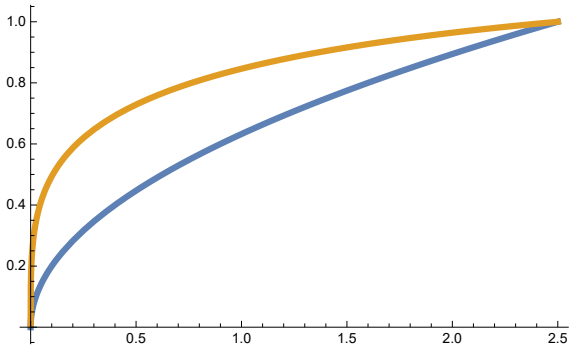

Figure 6.19: OPT and Online strategies for $t = 0.312$, $z = 0.2$

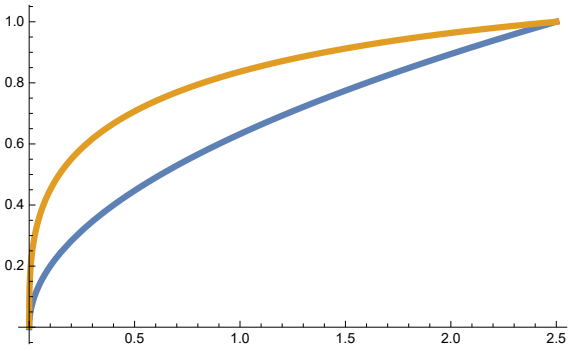

Figure 6.20: OPT and Online strategies for $t = 0.312$, $z = 0.3$

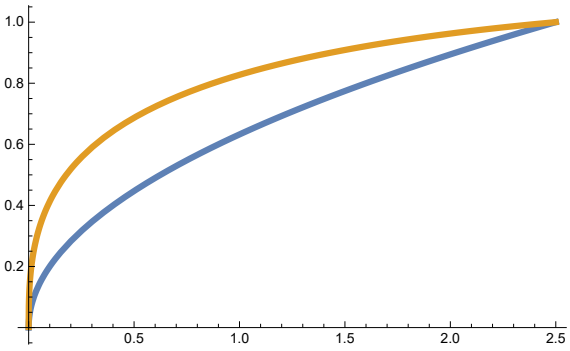

Figure 6.21: OPT and Online strategies for $t = 0.312$, $z = 0.4$

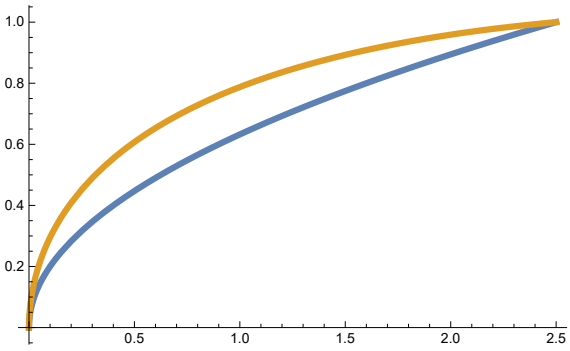

Figure 6.22: OPT and Online strategies for $t = 0.312$, $z = 0.9$

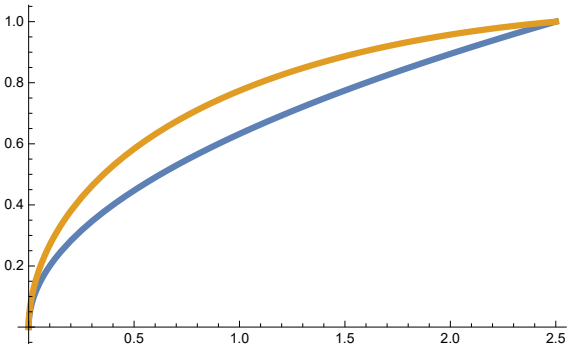

Figure 6.23: OPT and Online strategies for $t = 0.312$, $z = 1.1$

Similar to our earlier experiments, when z increases the online strategy is similar to the offline algorithm strategy. However, the z does not influence the strategy to change as much as the t value, in other words, the value of z needs to be increased by a larger amount than the t value to notice a difference in the strategy. Let us observe the costs generated by the strategies.

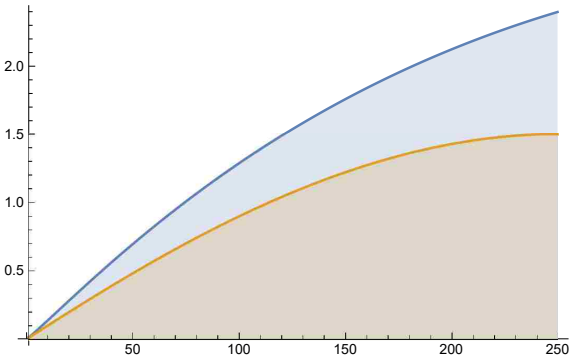

Figure 6.24: Cost of OPT and Online for $t = 0.312$, $z = 0.2$

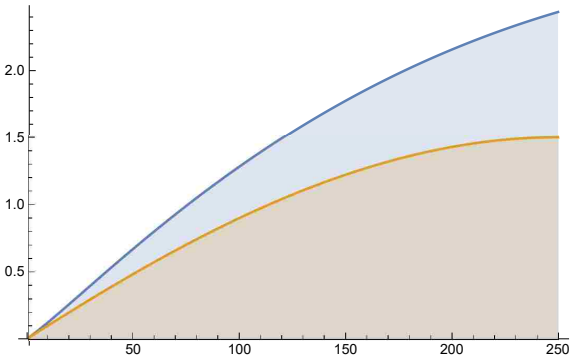

Figure 6.25: Cost of OPT and Online for $t = 0.312$, $z = 0.3$

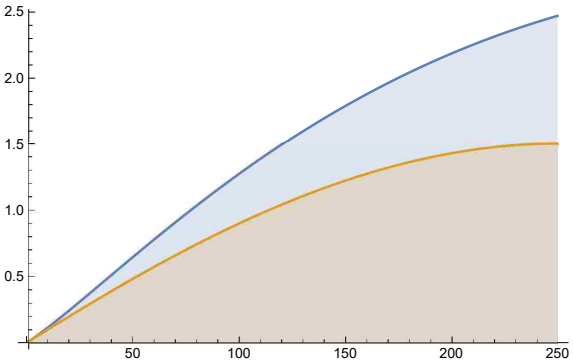

Figure 6.26: Cost of OPT and Online for $t = 0.312$, $z = 0.4$

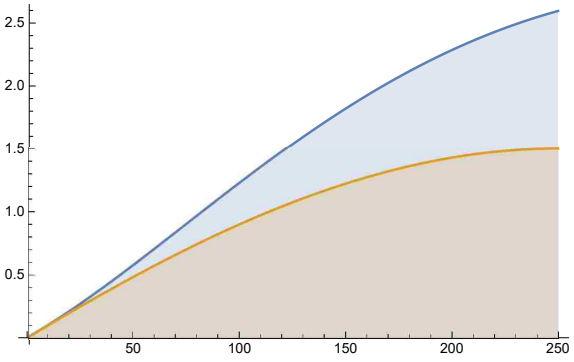

Figure 6.27: Cost of OPT and Online for $t = 0.312$, $z = 0.9$

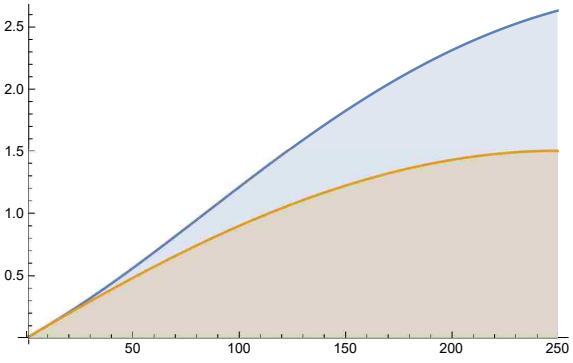

Figure 6.28: Cost of OPT and Online for $t = 0.312$, $z = 1.1$

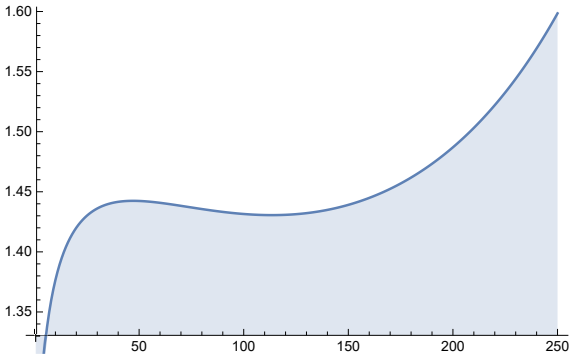

Figure 6.29: Competitive ratio $t = 0.312$, $z = 0.2$

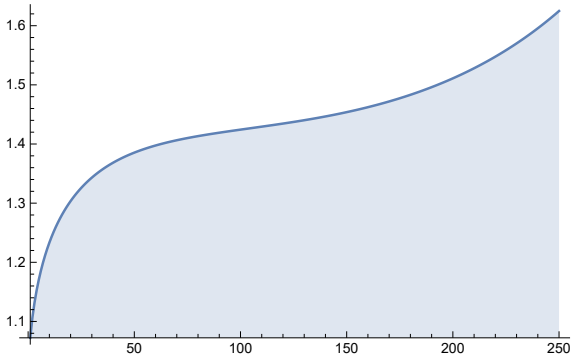

Figure 6.30: Competitive ratio $t = 0.312$, $z = 0.3$

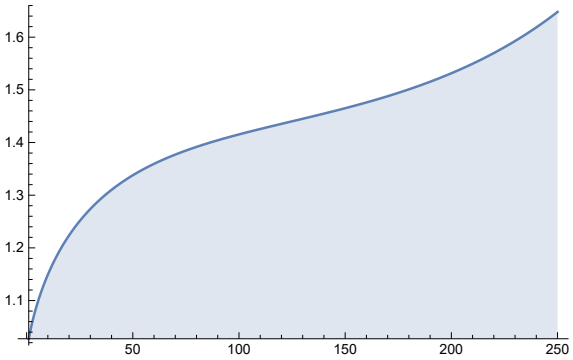

Figure 6.31: Competitive ratio $t = 0.312$, $z = 0.4$

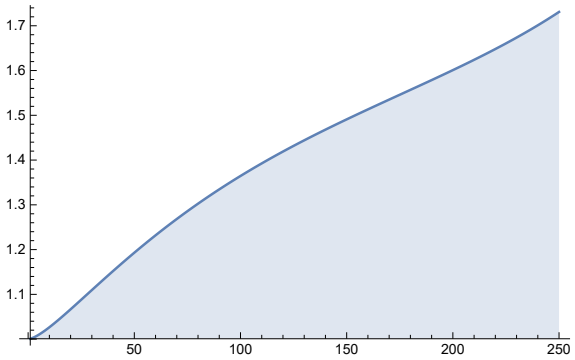

Figure 6.32: Competitive ratio $t = 0.312$, $z = 0.9$

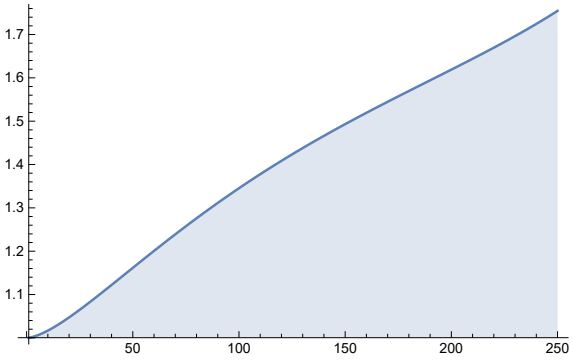

Figure 6.33: Competitive ratio $t = 0.312$, $z = 1.1$

Similar to the last experiment, as we increase z, the online and offline strategies are transitioning at the same rate to lower states, and the cost and competitive ratio is increasing and behaving similar to the lower envelope algorithm. We further analyze the continuous state machine for other functions for the online strategy, we have $\text{Strategy}_{\text{Linear}}(r) = r/x_m$, $\text{Strategy}_{\ln}(C, r) = \ln(Cr)/\ln(Cx_m + 1)$, and $\text{Strategy}_e(C, r) = (e^{Cr} - 1)/(e^{Cx_m} - 1)$.

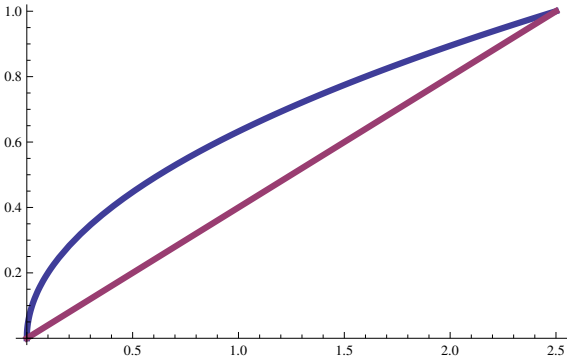

Figure 6.34: Strategy Linear Function

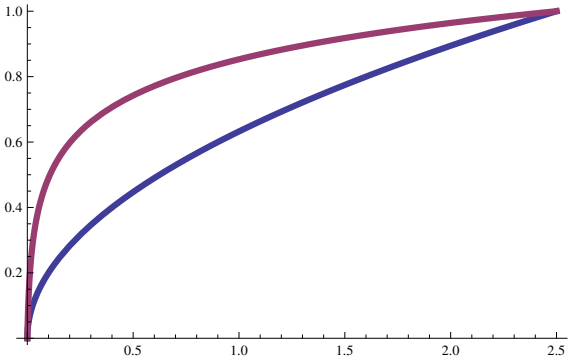

Figure 6.35: Strategy $\text{Strategy}_{\ln}(200, r)$

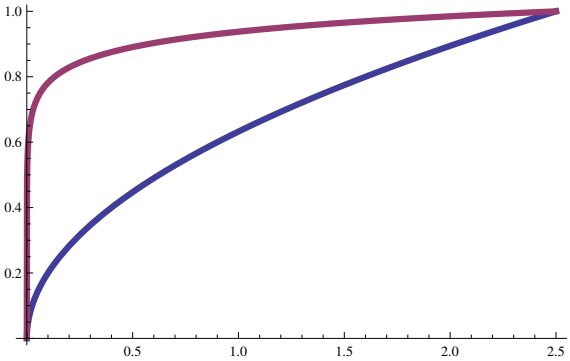

Figure 6.36: Strategy $\text{Strategy}_{\ln}(10^6, r)$

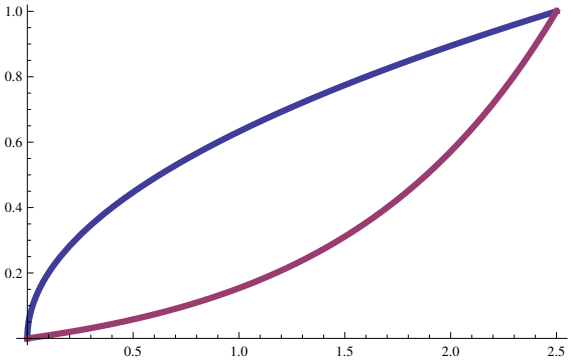

Figure 6.37: Strategy $\text{Strategy}_{e}(1, r)$

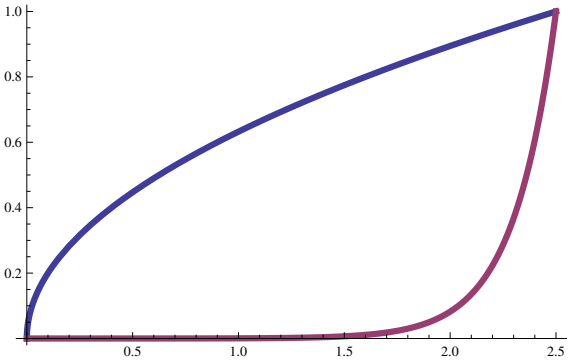

Figure 6.38: Strategy $\text{Strategy}_{e}(5, r)$

In the above figures, we have strategies, the linear function in Figure 6.34 and the exponential functions in Figures 6.37 and 6.38the online strategies are transitioning to the lower power states at slower rates than the offline algorithm, and Figures 6.35 and 6.36 we have strategies that transition to lower power states at faster rates than the offline algorithm, similar to the last experiment. Let us analyze the costs for the strategies.

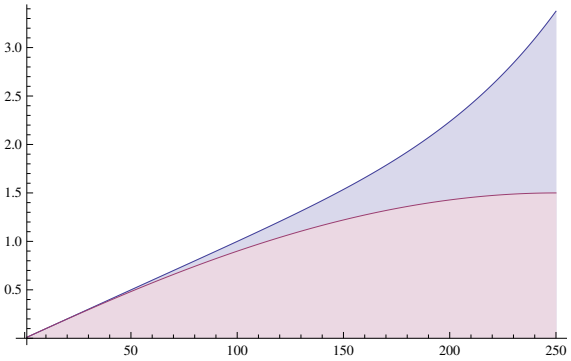

Figure 6.39: Cost Linear Function

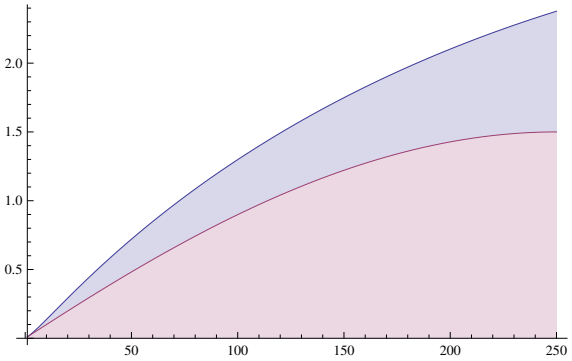

Figure 6.40: Cost Strategy$_{\ln}(200, r)$

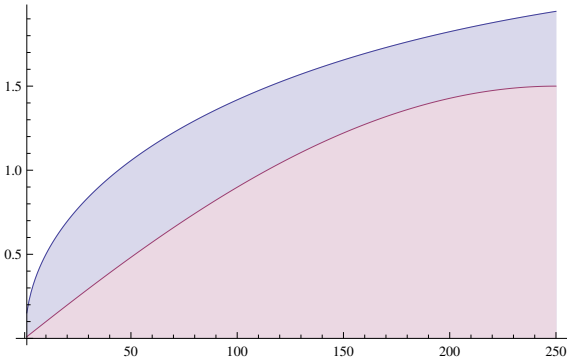

Figure 6.41: Cost Strategy$_{\ln}(10^6, r)$

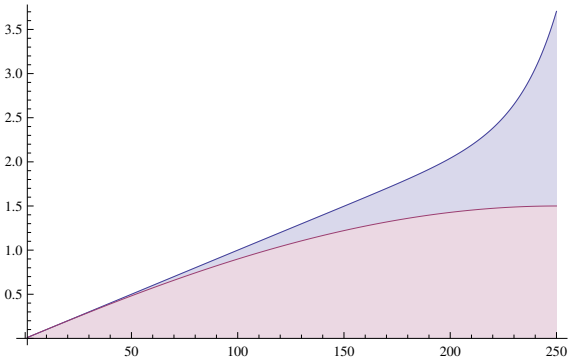

Figure 6.42: Cost Strategy$_e(1, r)$

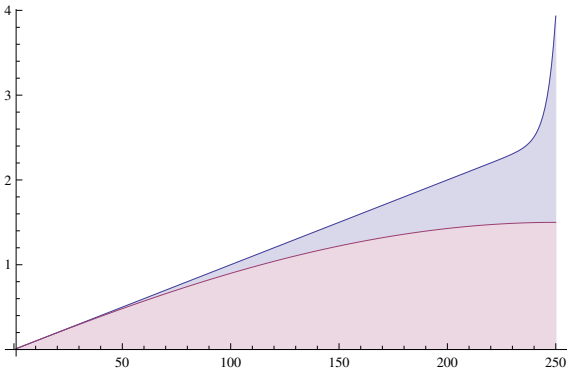

Figure 6.43: Cost Strategy$_e(5, r)$

We can see in this case, when the online strategies are transitioning to lower power states at a slower rate than the offline algorithm, the cost at the beginning of the idle period is smaller and as the idle time increases, the online costs are increasing, and their costs are larger than the strategies that are changing to lower power states at faster rates, in those strategies, the online cost is larger but does not increase the cost at such a rate as the strategies that are transitioning at a slower rate. Let us observe the competitive ratio of the strategies.

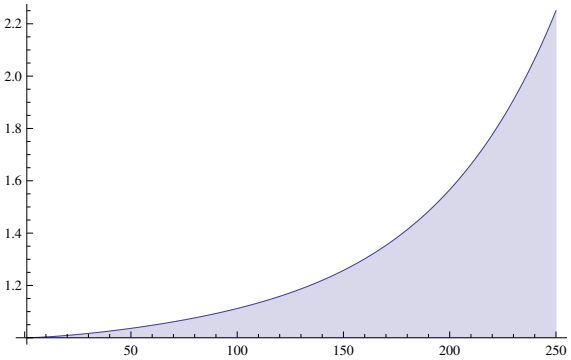

Figure 6.44: Competitive Ratio Linear Function

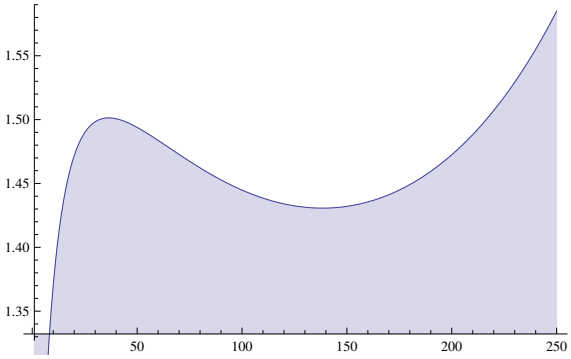

Figure 6.45: Competitive Ratio $\text{Strategy}_{\ln}(200, r)$

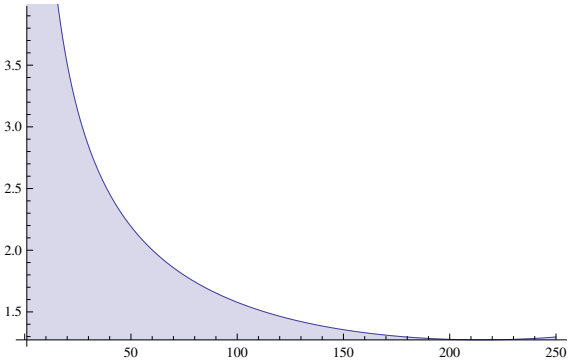

Figure 6.46: Competitive Ratio $\text{Strategy}_{\ln}(10^6, r)$

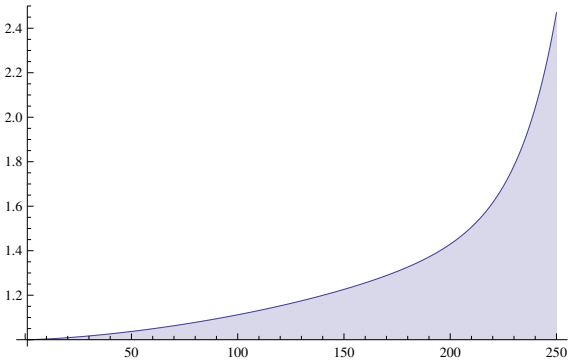

Figure 6.47: Competitive Ratio $\text{Strategy}_{e}(1, r)$

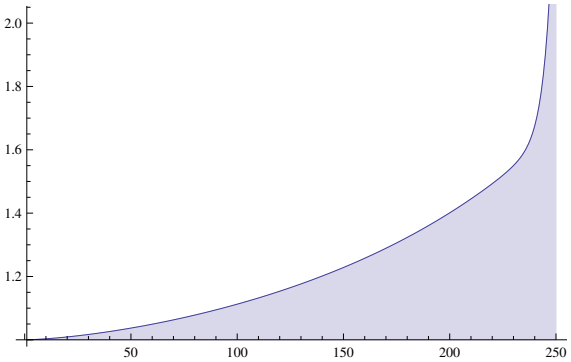

Figure 6.48: Competitive Ratio $\text{Strategy}_{e}(5, r)$

The competitive ratio curves show that the strategies that transition at slower rates have better competitive ratios than strategies that transition at a faster rate but at the end of the idle duration the faster transitioning strategies have better competitive ratios and thus their competitive ratios are overall smaller. The conclusion that we can see for the infinite state problem is that if we want to

have a lower competitive ratio, the online algorithm needs to use a strategy such that it transitions to lower power states at a faster rate than strategy used by the offline algorithm.

Chapter 7

The Decrease and Reset Algorithm

7.1 Details

In this chapter, we introduce the decrease and reset algorithm which allows the system to adjust its wait time based on how our system is used. For the two state power down problem, it is proven to be 2-competitive which is the minimal competitive ratio which we call OWCR which we saw in chapter 2. We introduce the decrease and reset algorithm (DRA) in our papers [8, 14] and another paper to be published titled Decrease and Reset for Power Down, to the Theoretical Computer Science journal. The concept of DRA is to increase the competitive ratio slightly by some $\epsilon > 0$, which allows us to generate a spectrum of algorithms. For instance if each request in σ arrive after β/α time units, the OWCR strategy obtains the worst case cost since it could have powered down earlier to save energy. Suppose if an algorithm would decrease its wait time after every request that arrives after β/α time units, which we will call a slack request, and we would increase the wait time if a request arrives at or before β/α, which is a busy request. The cost of that algorithm would be less than the cost of OWCR in that situation if it powers down earlier on a slack request. We have an infinite non-decreasing non-negative sequence x_1, x_2, x_3,... which are idle times that are assigned to the idle periods. The sequence of requests σ, which was introduced in chapter 2. The following piecewise function is used to determine which x_i will be used for the idle time after the i^{th} request.

$$f(i) = \begin{cases} f(i-1)+1 & \text{if } r_i - r_{i-1} \geq \beta/\alpha \text{ and } i \neq 1 \\ 1 & \text{otherwise} \end{cases}$$

So at request i, the wait time for the machine will be $x_{f(i)}$ and $f(i)$ will select the wait time from the infinite sequence. The behavior of DRA is after a request has been processed by the machine, if that request is a slack request, the wait time used for the next request is the next x_i in the sequence,

which decreases the wait time for the next request. This pattern continues as long as each request is a slack request. However, if a request is a busy request, then the wait time gets adjusted to the first value in the infinite sequence x_1. For the remainder of the chapter, $\alpha = 1$ and for most of the analysis we will also have $\beta = 1$ just to normalize the maximum wait time to β or 1 respectively, to simplify the equations.

Lemma 7.1.1. $\mathcal{R}_{\mathcal{DRA}} \leq 1 + \beta/x_1$.

Proof. For a given sequence $\sigma = r_1, r_2, r_3, ..., r_m$, we assume that all of these requests arrive after x_1 time units between each other. It is simple to see that the competitive ratio for m requests will be:

$$\mathcal{R}_{\mathcal{DRA}}(\sigma) \leq \frac{m(x_1 + \beta)}{mx_1} = 1 + \frac{\beta}{x_1} \tag{7.1}$$

In this case since every request arrives after x_1 time units, DRA will never decrease its wait time, it will never switch to x_i where $i > 1$ since $x_1 < \beta/\alpha$. In the offline model it is known that each request arrives exactly after x_1 time units so the machine will never power down, and hence the above competitive ratio is achieved. It is clear that the value of x_1 determines the value of the upper bound. If $x_1 = \beta/\alpha$, then this instance of the DRA is OWCR with a competitive ratio of 2. □

Lemma 7.1.2. *The worst case cost for DRA occurs when the last request is a busy request*

Proof. First we will examine the competitive ratio for one request. The online cost will always be $x_1 + \beta$. For a busy request the offline cost will be x_1 and β for a slack request. So the competitive ratio for a busy request will be $1 + \frac{\beta}{x_1}$ and for a slack request will be $1 + \frac{x_1}{\beta}$. If $x_1 = \beta$ then both would yield a 2-competitive upper bound but since $\epsilon > 0$, $x_1 < \beta$. Therefore for any possible x_1 value, $\frac{x_1}{\beta} < \frac{\beta}{x_1}$. Now we consider two identical blocks of length $l-1$ and then we will branch off two directions, one where the l^{th} request will be a busy request and the other will be a slack request. So the two blocks will be identical except for the last request, so only the last request needs to be examined. The cost of the online and offline algorithms before request l will be $Cost_{\text{online}}$ and $Cost_{\text{offline}}$ respectively. So the competitive ratio if the l^{th} is busy and slack will be

$$CR_{\text{busy}} = \frac{Cost_{\text{online}} + x_l + \beta}{Cost_{\text{offline}} + x_l}$$

$$CR_{\text{slack}} = \frac{Cost_{\text{online}} + x_l + \beta}{Cost_{\text{offline}} + \beta}$$

Notice that the online cost never changes no matter the type of the last request, busy or slack. As for the offline cost, $x_l < \beta$, so the offline cost is less when the l^{th} request is busy making the ratio larger, and when the last request is slack, the offline cost is β more which is larger than x_l which makes the ratio smaller compared to the busy competitive ratio. Which means the competitive ratio is maximized when the last request is busy in any given block. □

Lemma 7.1.3. *For an integer k,* $\mathcal{R}_{\mathcal{DRA}} \leq \frac{k\beta+\sum_{i=1}^{k} x_i}{(k-1)\beta+x_k}$

Proof. From lemma 7.1.2, the worst case is when the last request is a busy request. We will divide each block in which the k^{th} request is busy and all $k-1$ requests will be slack. The online cost will be simply $\sum_{i=1}^{k}(x_i+\beta) = k\beta + \sum_{i=1}^{k} x_i$. The offline cost will be $(k-1)\beta + x_k$ since in the offline sense, the machine will power down immediately after each request before k and only at the k^{th} request will the machine stay on for x_k time units. There can be up to m blocks, or in other words, m busy requests.

$$\mathcal{R}_{\mathcal{DRA}} \leq \frac{m\Big(k\beta + \sum_{i=1}^{k} x_i)\Big)}{m\Big((k-1)\beta + x_k)\Big)} = \frac{k\beta + \sum_{i=1}^{k} x_i}{(k-1)\beta + x_k} \tag{7.2}$$

□

We can set equation 7.2 to $2+\epsilon$ to compute the values of x_i.

$$\frac{k\beta + \sum_{i=1}^{k} x_i}{(k-1)\beta + x_k} \leq 2+\epsilon \tag{7.3}$$

If we solve for x_k, we have

$$x_k \geq \frac{1}{(1+\epsilon)}\left((2+\epsilon)\beta + \sum_{i=1}^{k-1} x_i\right) - k\beta$$

As one can see, this will be a recurrence relation. We can use elementary induction to obtain a closed form to compute any x_k value. In order to get a closed form we will rewrite equation 7.3 in the following way:

$$k\beta + \sum_{i=1}^{k} \geq (2+\epsilon)(k-1)\beta + x_k(2+\epsilon) \tag{7.4}$$

We can substitute $k+1$ for k in equation 7.4 and take the difference from 7.4 to obtain:

$$\beta + x_k \geq (2+\epsilon)\beta + (2+\epsilon)(x_k - x_{k-1})$$

Now we can reach the following recurrence:

$$x_k \geq \left(\frac{2+\epsilon}{1+\epsilon}\right)x_{k-1} - \beta$$

The recurrence can be solved to reach

$$x_k \geq -\epsilon\left(\frac{2+\epsilon}{1+\epsilon}\right)^k \beta + (1+\epsilon)\beta \tag{7.5}$$

As long as all x_i satisfy equation 7.5, then DRA will be $(2+\epsilon)$-competitive. If we let ϵ be an arbitrarily small constant ($\epsilon < 0.1$), the competitive ratio will not be much worse than that of the

OWCR. The real power of the DRA is that we can taper down the wait times if we have slack requests without increasing the competitive ratio by much. This allows us to save power without increasing the competitive ratio by much, since power is saved after each request since the wait times iteratively become smaller for slack requests. In [14], we introduce the term slack degree which is used to compute the competitive ratio. For each request in σ we can count all the busy and slack requests which will be denoted as $b(\sigma)$ and $s(\sigma)$ respectively. The slack degree $d(\sigma)$ will simply be the ratio between slack and busy requests, $d(\sigma) = s(\sigma)/b(\sigma) \geq d$ $(b(\sigma) \neq 0)$. The slack degree $d(\sigma)$ is the maximum d value that satisfies $s(\sigma)/b(\sigma)$.

Lemma 7.1.4. *The competitive ratio for DRA for long input σ with a slack degree $d > 0$ and with x_i values satisfying inequality 7.5 will be*

$$\mathcal{R}_{\mathcal{DRA}}(\sigma) \leq 1 + \frac{1}{d} + \frac{\sum_{i=1}^{\infty} x_i}{d\beta}$$

Proof. We first will have n sequences. The sequences will be labeled as σ_1, σ_2,...,σ_n. For each block σ_i, there will be $i-1$ slack requests and the i^{th} request, the last request of the block, will a busy request. So the online cost for σ_1, σ_2,...,σ_n will be $\beta + \sum_{i=1}^{n}\{\sum_{j=1}^{f(i)}(x_i + \beta)\} + x_1$. One can see that $f(i) = i$ for every i. The offline algorithm will power off for every slack request and only remain idle for the last request in the block, this behavior will occur for each block, so the offline cost will be $\beta + \sum_{i=1}^{n} x_{f(i)} + \sum_{i=1}^{n}(f(i)-1)\beta$

$$\mathcal{R}_{\mathcal{DRA}}(\sigma_1, \sigma_2, ..., \sigma_n) = \frac{\beta + \sum_{i=1}^{n}\{\sum_{j=1}^{f(i)}(x_i + \beta)\} + x_1}{\beta + \sum_{i=1}^{n} x_{f(i)} + \sum_{i=1}^{n}(f(i)-1)\beta}$$

$$\leq \frac{\beta + \sum_{i=1}^{n}\sum_{j=1}^{f(i)-1} x_j + \sum_{i=1}^{n} f(i)\beta + x_1}{\beta + \sum_{i=1}^{n}(f(i)-1)\beta}$$

$$\leq \frac{\beta + \sum_{i=1}^{n} x_i + \sum_{i=1}^{n}(f(i)-1)\beta + \beta n + x_1}{\beta + \sum_{i=1}^{n}(f(i)-1)\beta}$$

Since each block ends with a busy request, there will be n busy requests and $\sum_{i=1}^{n}(f(i)-1)$ represents the number of slack request, and thus $\sum_{i=1}^{n}(f(i)-1)/n = s(\sigma)/b(\sigma) \geq d$, after applying this substitution we have:

$$\leq \frac{\beta + n\sum_{i=1}^{n} x_i + nd\beta + \beta n + x_1}{\beta + nd\beta}$$

$$\leq 1 + \frac{\sum_{i=1}^{n} x_i + \beta + x_1/n}{d\beta + \beta/n}$$

Now we can set $n \to \infty$ and we have

$$\mathcal{R}_{DRA} = 1 + \frac{1}{d} + \frac{\sum_{i=1}^{\infty} x_i}{\beta d}$$

□

Theorem 7.1.5. *The competitive ratio of DRA is*

$$\mathcal{R}_{\mathcal{DRA}} = \min\left\{1 + \frac{1}{d} + \frac{\sum_{i=1}^{\infty} x_i}{d\beta}, 2 + \epsilon\right\}$$

Proof. From equation 7.5, if we set each delay time to

$$x_i = \begin{cases} -\epsilon\left(\frac{2+\epsilon}{1+\epsilon}\right)^i \beta + (1+\epsilon)\beta & \text{if } x_i > 0 \\ 0 & \text{otherwise} \end{cases}$$

The taper down values for x_i will cause the DRA to be $2 + \epsilon$ competitive since the x_i values were derived by setting the competitive ratio to $2 + \epsilon$ as shown in inequality 7.3. Lemma 7.1.4, shows the competitive ratio if the system is d-slack. And so, there are two possible competitive ratios, if a busy request occurs in the sequence, the competitive ratio will be $2 + \epsilon$ from 7.2 and if the system is d-slack, lemma 7.1.4 shows the competitive ratio for that situation. □

7.2 Budget Based Taper Down Algorithm for Two State Machine

We will see this approach for the three state machine in later chapters, here we consider an alternative approach to taper down the delay times using a budget. As we know, for the two state power down problem, the best algorithm we can have is 2-competitive. If we use a budget to taper down, we get the following equation

$$\frac{\alpha \cdot x_{\text{budget}} + \beta - b}{x_{\text{budget}}} = 2(1+\epsilon) \tag{7.6}$$

$$x_{\text{budget}} = \frac{\beta - b}{\alpha(2\epsilon + 1)} \tag{7.7}$$

in equations 7.6 and 7.7, x_{budget} will be the taper down wait time and $x_{\text{budget}} \leq \beta/\alpha$ will always hold and thus the competitive ratio will be increased to $2(1 + \epsilon)$, similar to the DRA. Initially the budget will be set to 0, and the budget b is adjusted throughout the course of the execution of the algorithm. We can have a cost curve

$$Cost(x, \text{RequestDelayTime}) = \begin{cases} \alpha x & \text{if } x < \text{RequestDelayTime} \\ \alpha x + \beta & \text{if } x \geq \text{RequestDelayTime} \end{cases}$$

where x is the wait time the machine will stay idle and RequestDelayTime is the actual wait time between requests. We update the budget after each request

$$b = Cost(\beta/\alpha, \text{RequestDelayTime}) - Cost(x_{\text{budget}}, \text{RequestDelayTime})$$

We take the difference of the cost of using β/α as the wait time with x_{budget} as the wait time. The budget b is the energy saved when we tapered down, and if the budget increases the wait time decreases and when the budget decreases the wait time increases. If the value of $b < 0$, at some point, we adjust $b = 0$ which is the initial budget. If using x_{budget} yields some savings then we taper down, similarly to the DRA if the request is a slack request the algorithm will have a smaller delay time for the next request. If we have a slack system, when we have several requests that arrive after β/α, and we power down before β/α, we will save power. This is the goal of the DRA and this budget based taper down algorithm. We will show experimental results of the budget based algorithm and DRA for two state and three state machines in chapter 10.

Chapter 8

Three State Decrease and Reset Algorithm

As shown earlier, we analyzed the decrease and reset algorithm (DRA), and with the DRA, we saw the we could increase the competitive ratio slightly and taper down the wait times depending on the slackness of the system. We saw that if we have a third state that we called INT added to the 2 state system, we reached a better upper bound of 1.8 than that of the 2 state machine which was 2-competitive. In this chapter, we combine the concepts of the three state model with the DRA . Let $u_1, u_2, u_2, ...$ be an infinite sequence of standby times in the ON state and let $q_1, q_2, q_3, ...$ be an infinite sequence of standby times in the INT state. So u_i is the duration of the machine in the ON state and q_i is the duration in the INT state and after $u_i + q_i$ time units the machine switches to the OFF state. Since this will be a three state machine, there are two switch times when we have the worst case cost which occurs immediately at the switch times. So we have two equations that we will use to compute these times.

$$\frac{\sum_{i=1}^{k-1}(u_i + aq_i + 1) + u_k + d}{\sum_{i=1}^{k-1} 1 + u_k} = CR + \epsilon \tag{8.1}$$

$$\frac{\sum_{i=1}^{k-1}(u_i + aq_i + 1) + u_k + aq_k + 1}{\sum_{i=1}^{k-1} 1 + u_k + q_k} = CR + \epsilon \tag{8.2}$$

Equation (8.1) is used to compute u_k after $k-1$ slack requests, and that u_k value is used to compute q_k in equation (8.2). In the two equations, it is assumed that for state INT, $a + d \geq 1$, which can be seen by the offline cost since constants a and d are absent in the offline cost. Similarly to the 2 state DRA, we increase the competitive ratio by a small constant ϵ, and we attempt to taper the standby

duration of the ON and INT state after every slack request, a request that occurs after $u_i + q_i$, and the durations reset back to u_1 and q_1 otherwise. So if we solve for u_k and q_k, we get

$$u_k = \max\left\{\frac{r(k-1) - \sum_{i=1}^{k-1}(u_i + aq_i + 1) - d}{1-r}, 0\right\} \tag{8.3}$$

$$q_k = \max\left\{\frac{r(k-1) + u_k(r-1) - \sum_{i=1}^{k-1}(u_i + aq_i + 1) - 1}{a-r}, 0\right\} \tag{8.4}$$

In (8.3) and (8.4), we substitute $r = CR + \epsilon$, so r will denote the increased competitive ratio which is the result of applying the DRA to a system. Since the value of u_k from (8.3) is needed to compute the value of q_k in (8.4), if we assume that $u_k > 0$, then we get:

$$q_k = \frac{1-d}{r-a} \tag{8.5}$$

So q_k does not taper down as long as $u_k > 0$. Initially, only the duration of the ON state will taper after each slack request while the duration of the machine in state INT does not change. When $u_k = 0$, several slack requests have arrived consecutively, and then the value of q_k will begin tapering down. We will obtain a closed form for u_k, as was done for the two state system. We can derive (8.1) to have

$$\sum_{i=1}^{k-1} u_i + a\sum_{i=1}^{k-1} q_i - (1-r)u_k = (r-1)(k-1) - d \tag{8.6}$$

Since we know that if any $u_i > 0$, then the values for q_i will never taper down so we can substitute $q_i = \frac{1-d}{r-a}$ and then we have

$$\sum_{i=1}^{k} u_i - ru_k = (r-1)k - r + (r-ak)\frac{1-d}{r-a} \tag{8.7}$$

Now we can simply substitute $k = k - l$ in (8.7) to have

$$\sum_{i=1}^{k-1} u_i - ru_{k-1} = (r-1)(k-1) - r + (r-a(k-1))\frac{1-d}{r-a} \tag{8.8}$$

If we subtract (8.8) from (8.7) we have the following recurrence

$$(1-r)u_k + ru_{k-1} = (r-1) - a\frac{1-d}{r-a} \tag{8.9}$$

Let us substitute the right hand side to $\phi = (r-1) - a\frac{1-d}{r-a}$

$$u_k - \phi = \frac{r}{r-1}(u_{k-1} - \phi) \tag{8.10}$$

In (8.10), we can see that $(r)/(r-1)$ is multiplied k times so we can rewrite (8.10)

$$u_k - \phi = \left(\frac{r}{r-1}\right)^{k-1}(u_1 - \phi) \tag{8.11}$$

From equation (8.3) from the last section, we can compute $u_1 = d/(r-1)$, substitution that for u_1 we obtain

$$u_k = \max\left\{\left(\frac{r}{r-1}\right)^{k-1}(d/(r-1)-\phi)+\phi, 0\right\} \tag{8.12}$$

For all $i < l$ if $u_i > 0$, then we know that q_i does not change, let l be an index such that $u_l = 0$ and $u_{l-1} > 0$, which can be computed from (8.12), so the index l is the first instance such that the duration becomes zero, in which this case the value of $q_l \leq q_{l-1}$ and this holds for all indices larger than l. So we will derive a formula for the tapering values for q_k, similar to how we derived the formula for u_k, we will start by deriving (8.2) by replacing k with l

$$\sum_{i=1}^{l} u_i + a\sum_{i=1}^{l} q_i - ru_l - rq_l = r(l-1) - l \tag{8.13}$$

As done earlier, we will substitute $l = l-1$

$$\sum_{i=1}^{l-1} u_i + a\sum_{i=1}^{l-1} q_i - ru_{l-1} - rq_{l-1} = r(l-2) - l + 1 \tag{8.14}$$

Once again we subtract (8.13) from (8.14)

$$u_l + aq_l + r(u_{l-1} - u_l) + r(q_{l-1} - q_l) = r - 1 \tag{8.15}$$

So as we assumed before, $u_l = 0$ so we make that substitution into (8.15)

$$aq_l + ru_{l-1} + r(q_{l-1} - q_l) = r - 1$$

$$q_l = \frac{r - 1 - r(u_{l-1} + q_{l-1})}{a - r} \tag{8.16}$$

This q_l from (8.16), will be used to solve the recurrence for q_l values. In order to derive this formula, we will begin with (8.15) which was obtained by subtracting (8.13) from (8.14). We will substitute l for t and $t \geq l$, so we have

$$aq_t + rq_{t-1} - rq_t = r - 1 \tag{8.17}$$

The difference between (8.17) and (8.15) is that in this case both $u_t = 0$ and $u_{t-1} = 0$ since $t > l$ in which at this point $u_l = 0$ and $u_{l+1} = 0$ must be true and $t > l$ so $u_t = 0$ and $u_{t-1} = 0$ must be true as well. Now when we solve for q_t, we have the following recurrence

$$q_t = \frac{r-1}{a-r} + \left(\frac{r}{r-a}\right)q_{t-1} \tag{8.18}$$

Let us substitute $\chi = \frac{r}{r-a}$, when we solve the recurrence we have

$$q_t = \max\left\{\chi^{t-l}q_l + \left(\frac{r-1}{a-r}\right)\left(\frac{\chi^{t-l}-1}{\chi-1}\right), 0\right\} \tag{8.19}$$

$\epsilon = 0.001$		$\epsilon = 0.01$		$\epsilon = 0.1$	
u	q	u	q	u	q
0.4993	0.4996	0.4938	0.4959	0.4444	0.4615
0.4970	0.4996	0.4708	0.4959	0.2460	0.4615
0.4918	0.4996	0.4193	0.4959	0	0.3417
0.4799	0.4996	0.3044	0.4959	0	0
0.4534	0.4996	0.0474	0.4959		
0.3936	0.4996	0	0.1433		
0.2591	0.4996	0	0		
0	0.4708				
0	0.0390				
0	0				

Table 8.1: Three State Taper Down Values for $a = 0.6$ and $d = 0.4$, CR = 1.8

$\epsilon = 0.001$		$\epsilon = 0.01$		$\epsilon = 0.1$	
u	q	u	q	u	q
0.9450	0.0540	0.9361	0.0537	0.8560	0.0512
0.9430	0.0540	0.9155	0.0537	0.6751	0.0512
0.9386	0.0540	0.8733	0.0537	0.3220	0.0512
0.9298	0.0540	0.7873	0.0537	0	0
0.9117	0.0540	0.6118	0.0537		
0.8747	0.0540	0.2538	0.0537		
0.7988	0.0540	0	0		
0.6433	0.0540				
0.3244	0.0540				
0	0				

Table 8.2: Three State Taper Down Values for $a = 0.1$ and $d = 0.9$, CR = 1.951

In table 8.1, we see the taper down values after every slack request for various ϵ values. As already stated, when the q values are above 0, the values for q do not change, once the value for u becomes 0, the value of q becomes 0 after only 1 extra slack request. When $\epsilon = 0.001$ it takes two extra slack requests for q to become 0, since ϵ is arbitrarily small. We use $a = 0.6$ and $d = 0.4$

because this produces the optimal competitive ratio for the case where $a + d = 1$. Let us examine other a and d values as well. We can see in table 8.2, that when the value of a is closer to 0, the values for q taper down to 0 at a more rapid pace once u becomes 0. In the previous table, it would take q one or two steps to reach 0, once u becomes 0. Here once u becomes 0, q simultaneously becomes 0. Let us look at a case when a is closer to 1 and d is closer to 0.

$\epsilon = 0.001$		$\epsilon = 0.01$		$\epsilon = 0.1$	
u	q	u	q	u	q
0.1097	0.8893	0.1086	0.8815	0.0989	0.8101
0.1076	0.8893	0.0879	0.8815	0	0.7354
0.1032	0.8893	0.0448	0.8815	0	0.4212
0.0939	0.8893	0	0.8408	0	0
0.0746	0.8893	0	0.6799		
0.0341	0.8893	0	0.3771		
0	0.8436	0	0		
0	0.6926				
0	0.4073				
0	0				

Table 8.3: Three State Taper Down Values for $a = 0.9$ and $d = 0.1$, CR = 1.911

Here in table 8.3, we see that the values of q taper down to 0 after 4 extra slack requests for smaller ϵ values to reach 0. Based on these 3 tables, we can see a trend that when a is arbitrarily larger, the values of q taper down nicely, i.e. takes more steps to taper down to 0, compared to when a is arbitarily smaller.

Chapter 9

Three State Problem with Reduced Delay Times

9.1 Analysis for Arbitrary Reduced Delay Times

In this chapter we develop a tapering down approach for the three state system which uses a budget as seen in chapter 2. We first demonstrate a few techniques to taper the wait times down in such a way where we decrease both the ON state duration as well as the INT state duration simultaneously, and show how the competitive ratio adjusts, which will lead to a budget based approach. We decrease x_1 and x_2 by some factor c, so the adjusted wait times are $(1-c)x_1$ and $(1-c)x_2$. Unlike with DRA, the durations of the machine in the ON state and INT state will decrease, instead of the duration of the ON state decreasing while the duration of the INT state remains unchanged unless the duration in the ON state becomes 0. But with this taper down model of the three state problem, we have the two competitive ratios CR'_1 and CR'_2 which denote the competitive ratios at $(1-c)x_1$ and $(1-c)x_2$ respectively. When we decrease the wait times, x_1 and x_2, the competitive ratios will increase.

Our goal is to have $CR'_1 = CR'_2$. The technique here will be to decrease x_1 and x_2 by $(1-c)$ and compute their respective CR'_1 and CR'_2 values. If we decrease x_1 and x_2 by $(1-c)$, we could situations where $CR'_1 \neq CR'_2$, which we know is not optimal, let us take a look at the following

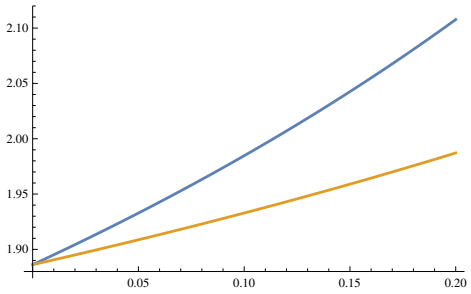

Figure 9.1: Competitive ratios when the wait times are decreased to $(1-c)x_1$ and $(1-c)x_2$

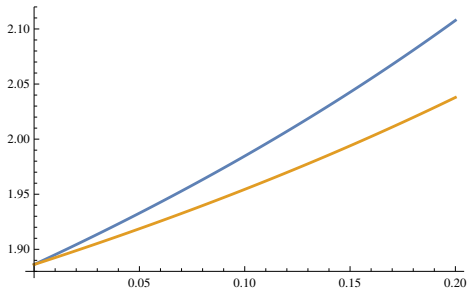

Figure 9.2: Competitive ratios when the wait times are decreased to $(1-c)x_1$ and $(1-1.3c)x_2$

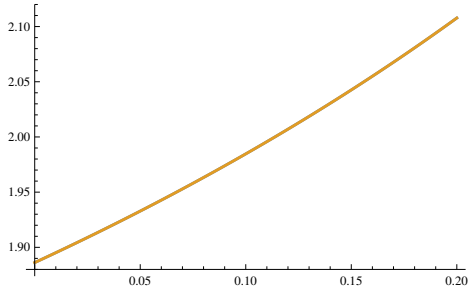

Figure 9.3: Competitive ratios when the wait times are decreased to $(1-c)x_1$ and $(1-1.7c)x_2$

Each of the figures show CR'_1 and CR'_2 as c increases. In Figures 9.1 and 9.2, $CR'_1 > CR'_2$. But it can also be seen that Figure 9.2 the overall competitive ratio does not increase since CR'_1 does not change and $CR'_2 < CR'_1$ is still true by decreasing x_2 by a larger factor than x_1, and furthermore, if x_2 decreases by a factor of $1.7c$ then $CR'_1 = CR'_2$ which is shown in figure 9.3. By decreasing the value of x_2 by a factor of $1.7c$, we can save power by powering down to the OFF state earlier while not increasing the overall competitive ratio since CR'_1 did not increase and CR'_2 does not become larger than CR'_1. In this particular case, the INT state standby and power up costs are $a = 0.45$ and $d = 0.3$. A search was applied to find this factor, we can apply a better approach to decrease x_1 and x_2 such that $CR'_1 = CR'_2$.

Suppose we have a factor c_1 to decrease the wait time x_1 and c_2 to decrease the wait time x_2. We will have the following competitive ratio:

$$CR'_1 = 1 + \frac{d}{x_1(1-c_1)}$$

$$CR'_2 = \frac{x_1(1-c_1) + a\big(x_2(1-c_2) - x_1(1-c_1)\big) + 1}{a\big(x_2(1-c_2)\big) + d}$$

Notice that the offline cost changes for CR_2 since we are decreasing the wait time for x_2 and $a\big(x_2(1-c_2) - x_1(1-c_1)\big) < 1$. We set the equations equal to each other and we can solve for c_2 which is:

$$c_2 = \frac{d^2 + x_1(c_1-1)(1+(a-1)(c_1-1)x_1) + d(x_1 - c_1x_1 + ax_2)}{adx_2}$$

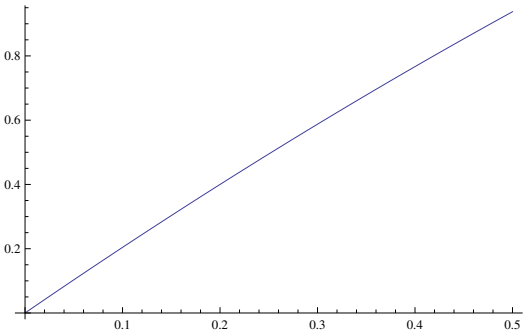

Figure 9.4: Increase in c_2 with respect to c_1, $a = 0.45$ and $d = 0.3$

We choose a value for c_1 and decrease x_1 to $(1-c_1)$ and if we use this c_1 value to compute c_2 and decrease x_2 to $(1-c_2)x_2$, we are guaranteed to have $CR'_1 = CR'_2$. Figure 9.4 shows that c_1 does not always equal c_2, and thus does not decrease x_1 and x_2 by the same rate in order for their competitive ratios to be equal.

9.2 Gains Obtained for Various c_1 Values

Using the c_1 and c_2 values, we decrease the wait times and this causes the competitive ratio increase. This section will focus on energy gained by choosing various c_1 and c_2 and not focus on competitive ratios. Let A be an online algorithm that does not decrease its wait times and let A' be an algorithm that adjusts the wait times to smaller values. So we can compute the gain obtained by A' by simply taking the difference, of the two algorithms with input sequence σ.

$$Power\ Gained = Cost_A(\sigma) - Cost_{A'}(\sigma)$$

So as long as the input sequence is not provided by the adversary we can have some savings when a new request arrives. This depends on delay time between requests and the values of c_1 and c_2, the following graphs show gains for various scenarios.

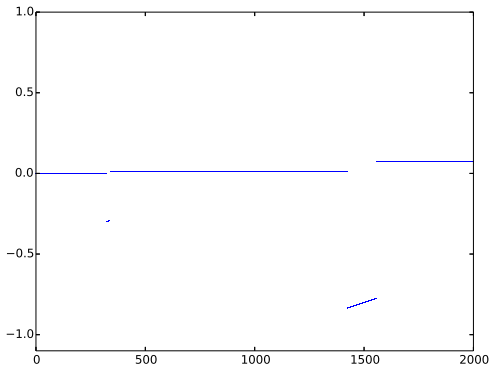

Figure 9.5: Gain for $c_1 = 0.05$

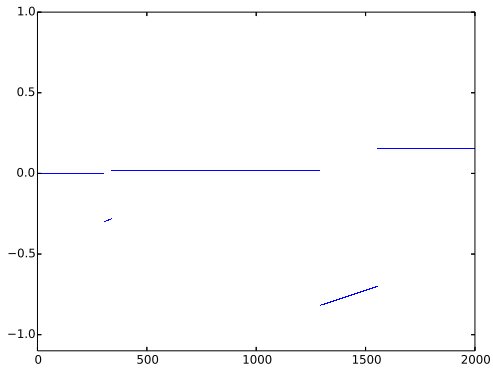

Figure 9.6: Gain for $c_1 = 0.10$

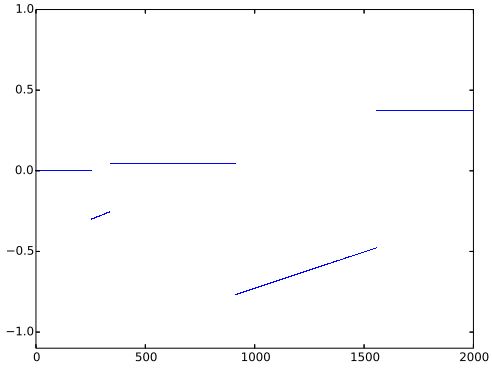

Figure 9.7: Gain for $c_1 = 0.25$

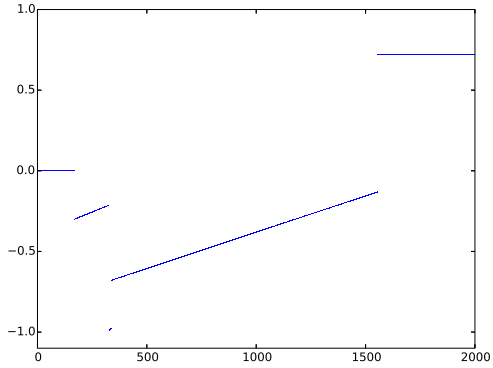

Figure 9.8: Gain for $c_1 = 0.50$

a	d	x_1	x_2	c_1	c_2	Adjusted x_1	Adjusted x_2
0.45	0.3	0.338528	1.55556	0.05	0.0856855	0.3216016	1.42227106
0.45	0.3	0.338528	1.55556	0.10	0.16987	0.3046752	1.29131702
0.45	0.3	0.338528	1.55556	0.25	0.41342	0.253896	0.91246038
0.45	0.3	0.338528	1.55556	0.5	0.789322	0.169264	0.32772227

Table 9.1: Adjusted times

Figures 9.5, 9.6, 9.7, and 9.8 show the gains for various c_1 and c_2 values with a range of possible request times. We can see that if we choose a smaller c_1 and subsequently a smaller c_2, the adjusted

wait times do not decrease as much and hence we do not have much savings. Even though we do not have a large gain, we do not have many moments of loss, in the general case, if a request arrives between $[(1-c_1), x_1]$ and $[(1-c_2)x_2, x_2]$ we see a loss. And in figure 9.1, since the adjusted wait times are not much less than the original x_1 and x_2 values, the loss duration is minimal compared to larger c_1 and c_2 values. When we have a larger c_1 value as shown in figure 9.8, we get a larger saving when a request arrives after x_2 compared to the other figures when we have a smaller c_1 value, however, for idle times from $(1-c_1)x_1$ to x_2 we lose energy, so we have a higher reward but with higher risk, when $c_1 = 0.05$ we have lower risk but with lower reward.

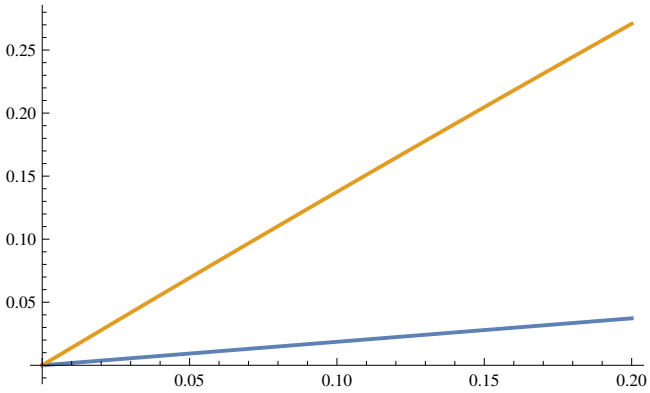

Figure 9.9: Gains for the adjusted wait times x_1 and x_2

In figure 9.9, we are computing the gains that we obtain with using the delay times x_1 and x_2 where the adjusted wait times give us a gain since the machine powered down to a lower power state before the original x_1 and x_2 times. Using these wait times, we see that when we increase c_1 we obtain larger gains. We obtain larger gains when the requests arrive at x_2 than when the requests arrive at time x_1. We can see that as c_1 increases the gain increases at a larger rate at time x_2 over x_1. This chooses c_1 up to 0.2 in which we still have some gain at time x_1.

So with this model we decrease the wait time in both durations, the duration in the ON state and INT state, which the 3 state DRA did not achieve. However here, when we increase c_1 the competitive ratio increases as well. The goal for the tapering down approach is to decrease the wait times while not increasing the competitive ratio, which the DRA does accomplish. So based on the last request, we want to compute a different wait time. The DRA computed a new wait time based on the value of ϵ and whether the next request was busy or slack, a slack request tapered down the wait time and a busy request reset the wait time. We will apply a similar model to the DRA to adjust the wait times rather than just choosing an arbitrary value for c_1 to adjust the wait time and competitive ratio.

9.3 Using a Budget to Compute Optimal Wait Times

We were able to compute new delay times using a possible range of values for c_1 and c_2, we now will use an alternative strategy to adjust the delay times. We will adjust the values for x_1 and x_2 based on the request times similarly to the DRA. This issue was well defined with DRA using ϵ and using this ϵ value, we could simply compute the wait times based on the number of consecutive slack requests. The goal here is to also given a set of requests σ, we would like to know what the wait time will be when the $(\sigma+1)^{th}$ arrives. For the model that has been analyzed in this chapter, we use a budget to determine the next wait time. When we think of a budget, we think of how much energy has been saved (if any) after σ requests and that will determine how long the idle time will be when $(\sigma+1)^{th}$ request arrives.

Initially the budget will be zero, at the beginning of the algorithm. Similarly to the DRA, we will increase the competitive ratio by some small constant, this will force the wait time to be slightly smaller than x_1 or x_2 at the start of the algorithm, when the budget is 0. We will denote y_1 and y_2 to represent the adjusted x_1 and x_2 times respectively.

$$\frac{y_1+d-b}{y_1+d} = CR(1+\epsilon)$$

$$y_1 = \max\left\{\frac{d-b}{CR(1+\epsilon)-1}, 0\right\} \tag{9.1}$$

Where CR is the competitive ratio for a 3 state system given a and d values, using equation 4.4. We will denote b as the budget. Similarly to the DRA, we increase the competitive ratio by a factor of $(1+\epsilon)$, so the idle time will initially be decreased. The idle time decreases as the budget increases, the budget will be adjusted based on the gain or loss we get after the most recent request arrives. The machine will switch to the INT state after y_1 time units. Calculating the value of y_2 is not as trivial as calculating y_1. There are two cases that need to be considered, if the machine powers down after time x_{opt_1} so $y_2 > x_{opt_1}$ or if the machine powers down before x_{opt_1} so $y_2 < x_{opt_1}$ which are likely scenarios if $a+d<1$ and in that case we must use a different optimal offline cost to compute y_2. The cost of the online algorithm will look the same in both instances, it will remain in the ON state for y_1 time units, and will stay in the INT state for y_2-y_1 time units and will have a power up cost from OFF to ON for when the request arrives. So its cost will be $y_1+a(y_2-y_1)+1$.

The cost of the offline algorithm will be $\min\{ay_2+d, y_2\}$. So based on the value of y_2, the offline algorithm will decide to stay idle for y_2 time units in the INT state or the ON state to handle the request. We will denote the two possible y_2 values as y_2' and y_2'' for each of the two cases. It can be seen that if $y_2 < x_{opt_1}$ then clearly the offline cost used to compute y_2 will y_2 time units in the ON

state, otherwise the offline cost will be $ay_2 + 1$. For y_2' we have the following:

$$\frac{y_1 + a(y_2' - y_1) + 1 - b}{ay_2' + d} = CR(1 + \epsilon)$$

$$y_2' = \max\left\{\frac{1 - b - dCR(1 + \epsilon) + y_1(1 - a)}{aCR(1 + \epsilon) - a}, 0\right\} \tag{9.2}$$

For y_2'' we have:

$$\frac{y_1 + a(y_2'' - y_1) + 1 - b}{y_2''} = CR(1 + \epsilon)$$

$$y_2'' = \max\left\{\frac{1 - b + y_1(1 - a)}{CR(1 + \epsilon) - a}, 0\right\} \tag{9.3}$$

In both cases like with y_1 from (9.1), when the wait times become negative, they get set to 0. The issue here is when y_2' and y_2'' is computed, it is not trivial as to which value is the appropriate value to choose for a given ϵ or budget value b. We have to be able to decide which y_2 value is correct, i.e. the y_2 which used the minimal offline cost to compute that value. The way we calculate the budget is to compute the gain or loss of using x_1 and x_2 transition times with these adjusted transition times y_1 and y_2. Let us define a piecewise function that computes the worst case online cost

$$OnlineCost(X_1, X_2, request) = \begin{cases} X_1 + d & \text{if request } \leq X_1 \\ X_1 + a(request - X_1) + d & X_1 < \text{ request } < X_2 \\ X_1 + a(X_2 - X_1) + 1 & \text{if request } \geq X_2 \end{cases}$$

where X_1 and X_2 will be the instant the machine transition from the on state to the intermediate state, and the intermediate state to the off state respectively. The parameter *request* denotes the next request time, so we can compute the gain

$$gain = OnlineCost(x_1, x_2, request) - OnlineCost(y_1, y_2, request)$$

The value of *gain* could be either positive or negative which denotes a gain or a loss and the we accumulate this to the budget. If the value of b ever becomes negative, we simply reset it back to 0, otherwise if we have some gain and the budget increases, then the value of y_1 and y_2 will be decreased.

Lemma 9.3.1. *Given a worst case transition time $\mathcal{Y}$, if $a\mathcal{Y} + d < \mathcal{Y}$, then $y_2' > y_2''$ must hold.*

Proof. We will first consider the two equations:

$$\frac{y_1 + a(y_2' - y_1) + 1 - b}{ay_2' + d} = CR(1 + \epsilon) \tag{9.4}$$

$$\frac{y_1 + a(y_2'' - y_1) + 1 - b}{y_2''} = CR(1+\epsilon) \tag{9.5}$$

We will consider the delay time $\mathcal{Y}$ such that it satisfies equation (9.5). By substituting y_2'' with $\mathcal{Y}$. If we substitute this $\mathcal{Y}$ into (9.4), we have the following:

$$\frac{y_1 + a(\mathcal{Y} - y_1) + 1 - \max\{b, 0\}}{a\mathcal{Y} + d} > CR(1+\epsilon) \tag{9.6}$$

Since the numerators in both (9.4) and (9.5) are identical, their values will be the same, and since $a\mathcal{Y} + d < \mathcal{Y}$, then the ratio in (9.6) substituting the formula with $\mathcal{Y}$, the ratio must be larger than $CR(1+\epsilon)$. We can rewrite (9.6) to:

$$\frac{y_1 + (a\mathcal{Y} - y_1) + 1 - \max\{b, 0\}}{a\mathcal{Y} + d} = \frac{y_1(1-a) + 1 - b - \max\{b, 0\}}{a\mathcal{Y} + d} + 1 > CR(1+\epsilon) \tag{9.7}$$

Since we want to satisfy the above equation, we must choose another delay time which will be denoted by $\mathcal{Y}'$. From (9.7), it is trivial that we must increase the value for $\mathcal{Y}$ to decrease its ratio until the ratio equals $CR(1+\epsilon)$, so $\mathcal{Y}' > \mathcal{Y}$. Therefore if we choose y_2' and y_2'' that satisfy their respective equations and if $ay_2' + d > y_2''$, then $y_2' > y_2''$ must hold. □

Lemma 9.3.2. *Given a worst case transition time $\mathcal{Y}$, if $a\mathcal{Y} + d > \mathcal{Y}$, then $y_2' < y_2''$ must hold.*

Proof. As with lemma 9.3.1, we will use the following equations:

$$\frac{y_1 + a(y_2' - y_1) + 1 - \max\{b, 0\}}{ay_2' + d} = CR(1+\epsilon) \tag{9.8}$$

$$\frac{y_1 + a(y_2'' - y_1) + 1 - \max\{b, 0\}}{y_2''} = CR(1+\epsilon) \tag{9.9}$$

Once again, we choose a value $\mathcal{Y}$ that satisfies (9.9), by substituting y_2'' with $\mathcal{Y}$. We will now substitute y_2' with $\mathcal{Y}$ in (9.8), to have:

$$\frac{y_1 + (a\mathcal{Y} - y_1) + 1 - \max\{b, 0\}}{a\mathcal{Y} + d} = \frac{y_1(1-a) + 1 - \max\{b, 0\} - d}{a\mathcal{Y} + d} + 1 < CR(1+\epsilon) \tag{9.10}$$

Since our initial assumption was that $a\mathcal{Y} + d > \mathcal{Y}$, then the equation (9.10), and the derived form both have to be less than $CR(1+\epsilon)$, since once again substituting y_2' and y_2'' with $\mathcal{Y}$ will have identical numerators but the denominator in (9.10) will be larger based on our initial assumption so this ratio must be less than $CR(1+\epsilon)$. We again must choose a different value $\mathcal{Y}'$ such that (9.10) equals $CR(1+\epsilon)$. In order to increase the ratio, we need to pick a smaller value than $\mathcal{Y}$ so clearly $\mathcal{Y}' < \mathcal{Y}$ such that y_2' substituted with $\mathcal{Y}'$ that satisfies (9.9). So if we choose y_2' and y_2'' that satisfy their respective equations and if $ay_2' + d > y_2''$, then $y_2' < y_2''$ must hold. □

Lemma 9.3.3. *Given x_{opt_1}, for values y_2' and y_2'', unless $y_2' = y_2'' = x_{opt_1}$, they will both be greater than or less than x_{opt_1}.*

Proof. We need to consider a few cases for this proof. As noted before for some delay $\mathcal{Y}$, if $a\mathcal{Y}+d < \mathcal{Y}$, then $\mathcal{Y} > x_{opt_1}$ and if $a\mathcal{Y}+d > \mathcal{Y}$, then $\mathcal{Y} < x_{opt_1}$. For the first case, if $y_2'' > x_{opt_1}$, then $ay_2'+d < y_2''$ and according to lemma 9.3.1, $y_2' > y_2''$ and therefore $y_2' > x_{opt_1}$ so in this case $y_2' > y_2'' > x_{opt_1}$ so both y_2' and y_2'' are greater than x_{opt_1}. Now let $y_2'' < x_{opt_1}$ so therefore $ay_2' + d > y_2''$. In this case, lemma 9.3.2 proves that $y_2'' > y_2'$ must hold and thus $x_{opt_1} > y_2'' > y_2'$ and therefore both y_2' and y_2'' both are less than x_{opt_1}.

Those were the two trivial cases. Let us examine when $y_2' < x_{opt_1}$, so $ay_2' + d > y_2''$ and therefore we know that $y_2'' > y_2'$. Let us assume the contrary that we compute y_2'' and $y_2'' > y_2'$ still holds, but $y_2'' > x_{opt_1}$, so we have a scenario where $y_2'' > x_{opt_1} > y_2'$. This scenario can never happen because if $y_2'' > x_{opt_1}$ then $y_2' > y_2''$ must hold according to lemma 9.3.1, so this leads to a contradiction.

For the last case, let $y_2' > x_{opt_1}$ so $ay_2' + d < y_2''$ and therefore $y_2' > y_2''$. Let us say that when y_2'' is computed, we get a value $y_2'' < x_{opt_1}$ so then we have the scenario $y_2' > x_{opt_1} > y_2''$, when $y_2'' < x_{opt_1}$, this implies that $ay_2' + d > y_2''$ must hold, according to lemma 9.3.2, if $ay_2' + d > y_2''$ then $y_2'' > y_2'$, so in this case when we have $y_2' > x_{opt_1} > y_2''$, we reach a contradiction because lemma 9.3.2 proves this cannot happen. So in every case, unless y_2' and y_2'' equal x_{opt_1}, both y_2' and y_2'' have to be greater than or less than x_{opt_1} to avoid reaching a contradiction. □

From lemma 9.3.3, since $y_2' > y_2'' > x_{opt_1}$ or $y_2' < y_2'' < x_{opt_1}$ will always hold. It can be seen that if the parameters for INT state satisfy $a + d \geq 1$, then we know that $x_{opt_1} = x_{opt_2} = x_2$, so the maximum possible value for y_2' or y_2'' must be x_{opt_1} so only y_2'' will be considered for the adjusted x_2, and y_2' will never be considered. If $a + d < 1$, than for a small enough b, $y_2' > y_2'' > x_{opt_1}$ and for a large enough b, $y_2' < y_2'' < x_{opt_1}$, so obviously for a certain b value, the curves will cross, i.e. $y_2' = y_2'' = x_{opt_1}$, we can simply set equations (9.2) and (9.3) equal to each other and solve for b to get:

Theorem 9.3.4. *Given a budget b, the adjusted delay time y_2 for the online algorithm will be* $y_2 = \max\{y_2', y_2'', 0\}$

Proof. From lemma 9.3.3, we know that one of two scenarios are possible, either $y_2' > y_2'' > x_{opt_1}$ or $y_2' < y_2'' < x_{opt_1}$. If both y_2' and y_2'' are larger than x_{opt_1}, then lemma 9.3.1 shows that $y_2'' < y_2'$ and y_2' is computed using the optimal offline cost, and when both values are less than x_{opt_1} then lemma 9.3.2 shows that $y_2'' > y_2'$ and y_2'' uses the optimal offline cost to compute its value. So the maximum of the two will yield the correct standby time for y_2, and if they both become negative for a large enough budget b, then the standby time $y_2 = 0$. □

When we compute the value for y_1, there was just one equation, and this will represent the adjusted x_1 time. For y_2 there are two different values and we have to choose the appropriate value for an instance, the last theorem demonstrates a simple way to choose the appropriate value. As noted earlier, if the budget parameter increases, the delay times begin tapering down and taper back up if it decreases. If both y_2' and y_2'' are smaller than x_{opt_1}, then y_2'' will be used and will be the maximal value. Similarly to the DRA, the ϵ value determines the competitive ratio and also determines the initial tapered down value when $b = 0$, for DRA w_{x_1} denoted the initial taper down value. In this algorithm, the taper values are maximized when $b = 0$.

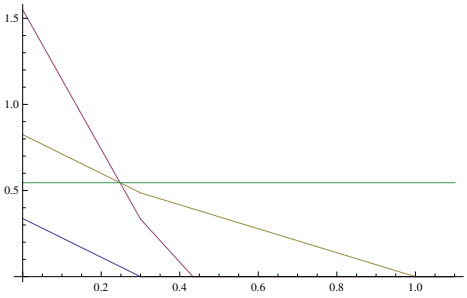

Figure 9.10: Taper down values with respect to b, $a = 0.45$, $d = 0.3$, $\epsilon = 0.001$

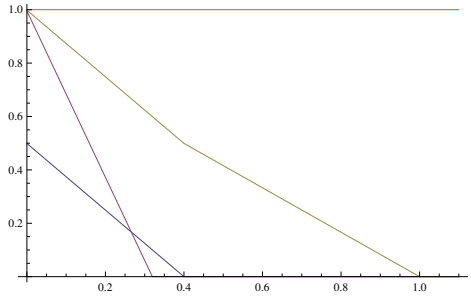

Figure 9.11: Taper down values with respect to b, $a = 0.6$, $d = 0.4$, $\epsilon = 0.001$

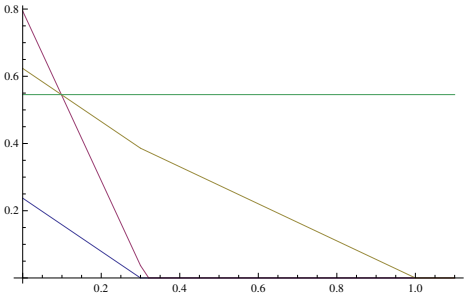

Figure 9.12: Taper down values with respect to b, $a = 0.45$, $d = 0.3$, $\epsilon = 0.1$

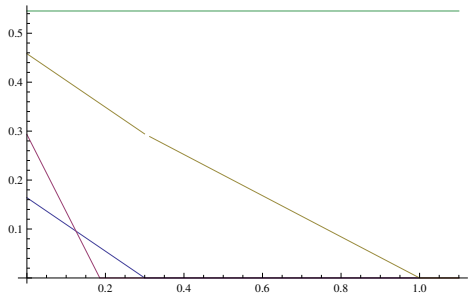

Figure 9.13: Taper down values with respect to b, $a = 0.45$, $d = 0.3$, $\epsilon = 0.5$

The above figures show the values of y_1, y_2', and y_2'', the values decrease as the budget increases. we can see that in figure 9.13, since $a + d \geq 1$, y_2'' is always greater than y_2', which we already know is the case. The rest of the figures demonstrate the adjusted wait times with a different ϵ value. The horizontal line in all the figures represent $x_{1_{OT}}$ which acts as a threshold as the which adjusted idle time we will use for y_2. Figures 9.10, 9.13, and 9.12 show the adjusted times with various ϵ values. The main observation to notice is that when ϵ becomes large enough, the maximum y_2' and y_2'' value will be less than x_{opt_1} and in this situation, the system will behave like a system in which $a + d \geq 1$

since both adjusted wait times are indeed less than x_{opt_1}.

9.4 Experimental Results

For our current 3 state taper down model, a script in python was written with all the parameters needed to correctly compute the y_1 and y_2 values. The input would take the duration in the idle state until the next request would arrive. A few test cases were used in which all cases had the same values for a, d, and ϵ. After each request the gain or loss for each request is accumulated to the budget b, and that would result in a possible new idle time y_1 and y_2. For the tables, we use $a = 0.4$, $d = 0.4$, $\epsilon = 0.01$, Competitive ratio = 1.9, the adjusted competitive ratio $CR(1 + \epsilon) = 1.919$, $x_1 = 1/3$, and $x_2 = 1.75$. These values will never change throughout the duration of the algorithm in all the input sets.

y_1	y_2	Budget	Gain	Next request duration
0.326441784548	1.68706493669	0.0	0.0	0.0
0.321942405581	1.66847256278	0.0041349292710	0.00413492927095	0.34
0.314505456015	1.63774153342	0.0109694859223	0.00683455665133	0.5
0.253351905393	1.38504228589	0.0671695989437	0.05620011302140	1.75
0.253351905393	1.38504228589	0.0671695989437	0.0	0.1
0.201133345258	1.16926428576	0.1151584557080	0.04798885676410	1.3
0.0	0.37737147186	0.4267727342470	0.31161427853900	2
0.0	0.0	1.1758241455000	0.74905141125700	2
0.0	0.41091234661	0.3758241455040	-0.8	0.2
0.321942405581	1.66847256278	0.0	-0.4999649386430	0.411

Table 9.2: Input set 1

We can see that if the idle duration, in any step, is in the range $x_1 \leq r < y_2$ and $r \geq x_2$, we have a gain, which subsequently causes the values of y_1 and y_2 to taper down after that request is processed in which after the system goes back into its idle period. In the DRA model, if $x_1 \leq r < y_2$ is when the request arrives, this request would be marked as a busy request which would cause the system to reset to its wait time in which the algorithm started. In this model, it creates a taper down. Also, if the request arrives before y_1, then there will be no gain in which no changes occur for y_1 and y_2 after that request. Once both $y_1 = y_2 = 0$, if the budget value b is not large enough, a busy request in which $r \to 0$, the loss can cause the entire budget to become 0 or a negative which

basically resets the y_1 and y_2 values which is depicted in the last row in table 9.2.

y_1	y_2	Budget	Gain	Next request duration
0.326441784548	1.68706493669	0.0	0.0	0.0
0.294549559746	1.55527960080	0.029308954593	0.029308954593	2
0.184475105088	1.10042895710	0.130467378424	0.101158423831	2
0.087288013742	0.69883159106	0.219782315371	0.089314936947	0.4
0.326441784548	1.68706493669	0.0	-0.551905444669	0.70

Table 9.3: Input set 2

Here we can see that the y_1 and y_2 values taper as usual since the nature of the delay times suggest that behavior. In the last row of table 9.3, we can see that when the request occurs right around time $\approx y_2$, the gain becomes negative and its which causes the budget to become negative and which causes the adjusted wait times to reset to their initial value.

y_1	y_2	Budget	Gain	Next request duration
0.326441784548	1.68706493669	0.0	0.0	0.0
0.294549559746	1.55527960080	0.029308954593	0.029308954593	2
0.184475105088	1.10042895710	0.130467378424	0.101158423831	2
0.087288013742	0.69883159106	0.219782315371	0.089314936947	0.4
0.326441784500	1.68706493670	0.0	-0.298372808245	0.09

Table 9.4: Input set 3

In table 9.4 and table 9.3, the inputs are identical in the first 4 rows, but in the last row we input $r \approx y_1$, which causes the algorithm to behave in a similar way to the results shown in table 9.3. This last input causes the gain to be negative and this causes the budget to be negative and the adjusted idle times y_1 and y_2 once again resets.

Chapter 10

Comparison with DRA and Budget Based Algorithm with OWCR

10.1 Slack Systems

In this chapter, we show experimental results that show the costs of the two state DRA, Budget Based technique, and OWCR against a random large input sequence generated using an even distribution. The DRA, Budget Based Algorithm, and OWCR were introduced earlier in the paper and it was proven that the optimal competitive ratio for the two state power down problem was 2-competitive, which is the competitive ratio of OWCR. We know that DRA is $(2+\epsilon)$-competitive in which ϵ is an arbitrarily small constant and the budget based algorithm is $2(1+\epsilon)$-competitive, so therefore the worst case cost of DRA and the budget based will be greater than OWCR. However this is always true when we have the worst case input sequence fed into the algorithms by the adversary. Here we will analyze when the inputs are random, we will analyze when the input sequences are slack, where slack degree $d > 1$, and when the inputs are busy, where slack degree $d < 1$. For the analysis, we have a set of inputs for each slack degree and the input length will be 100 requests.

Each input sequence was constructed using a random number generated using python. For the rest of this chapter, we will be generating several input sequences for various slack degrees and using various values of ϵ, for the DRA and the budget based approach, compare the three costs. The goal is to show that even though the tapering down approaches have a larger competitive ratio than OWCR, these techniques could have favorable results over OWCR, for certain types of input, when the inputs are not necessarily worst case inputs.

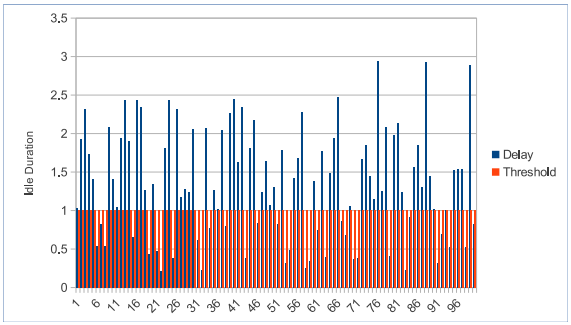

Figure 10.1: Slack degree d = 2, input set 1

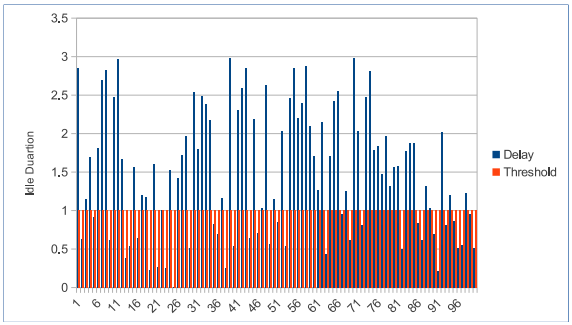

Figure 10.2: Slack degree d = 2, input set 2

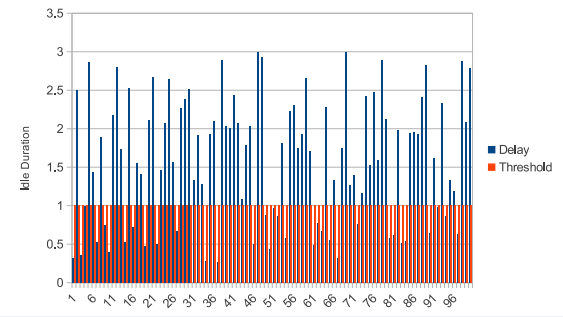

Figure 10.3: Slack degree d = 2, input set 3

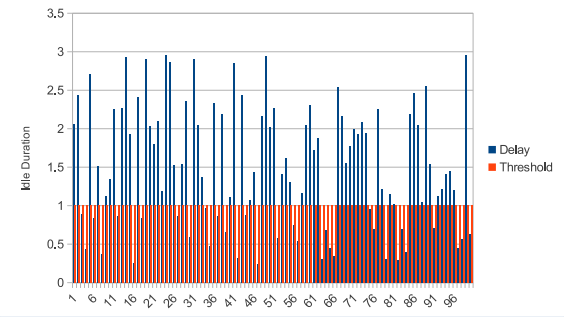

Figure 10.4: Slack degree d = 2, input set 4

In the above figures, the requests that are inside what appears to be a shaded region are busy requests and any request above that shaded region can be considered slack requests. For these examples, we will set $\beta = 1$ and $\alpha = 1$, so the power down threshold will be 1 idle time unit so a request at or before 1 will be a busy request and any request after 1 will be slack, this will be the case for all input sequences used in the remainder of this chapter.

DRA			Budget Based Algorithm			OWCR	OPT
$\epsilon = 0.001$	$\epsilon = 0.01$	$\epsilon = 0.1$	$\epsilon = 0.0005$	$\epsilon = 0.005$	$\epsilon = 0.05$		
151.08	149.98	132.57	107.88	113.52	102.77	151.81	84.81
149.44	143.20	134.29	120.54	109.72	103.92	150.55	85.55
153.77	150.46	135.44	106.06	109.55	109.63	153.92	86.93
153.49	147.33	131.53	124.02	104.30	105.71	153.66	86.66

Table 10.1: Costs for d = 2

In table 10.1, each row contains the costs for DRA, Budget based, OWCR, and OPT for each input sequences from figures 10.1, 10.2, 10.3, and 10.4 respectively. We can see that when ϵ is

smaller, DRA has a similar cost to OWCR, the same is true for the budget based algorithm. This is clearly due to the fact that DRA tapers down slower when ϵ is smaller, and thus it almost mimics the behavior of OWCR. However, when ϵ is larger, we see a significant savings for the DRA over OWCR, since the wait time for DRA will reach zero after fewer amount of consecutive slack requests. We will perform the same routine for slack degree 4, 6, and 8. We do not see that much of a difference with the budget based technique when the value of ϵ changes. However, we see that the budget based algorithm is more optimal compared to the DRA.

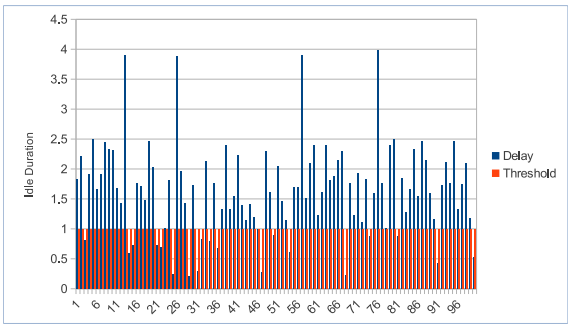

Figure 10.5: Slack degree d = 4, input set 1

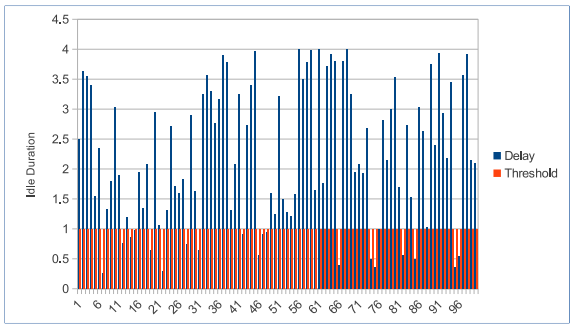

Figure 10.6: Slack degree d = 4, input set 2

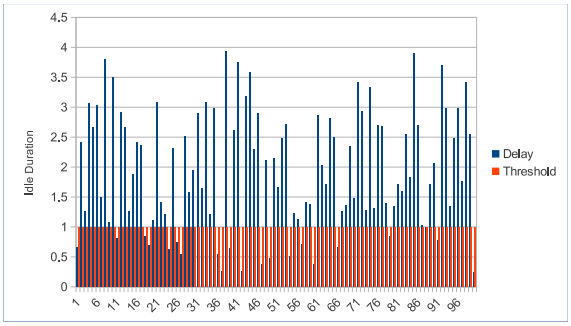

Figure 10.7: Slack degree d = 4, input set 3

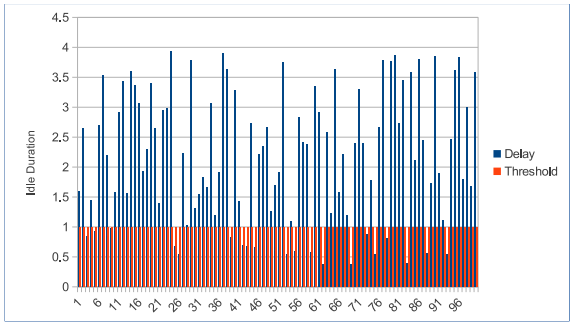

Figure 10.8: Slack degree d = 4, input set 4

DRA			Budget Based Algorithm			OWCR	OPT
$\epsilon = 0.001$	$\epsilon = 0.01$	$\epsilon = 0.1$	$\epsilon = 0.0005$	$\epsilon = 0.005$	$\epsilon = 0.05$		
165.20	148.32	126.42	108.79	105.58	105.18	172.27	92.27
163.80	151.88	130.76	108.45	105.02	102.77	172.71	92.71
163.80	147.78	132.08	117.94	105.43	102.43	171.57	91.57
161.77	152.32	132.08	115.63	113.33	108.11	173.08	93.03

Table 10.2: Costs for d = 4

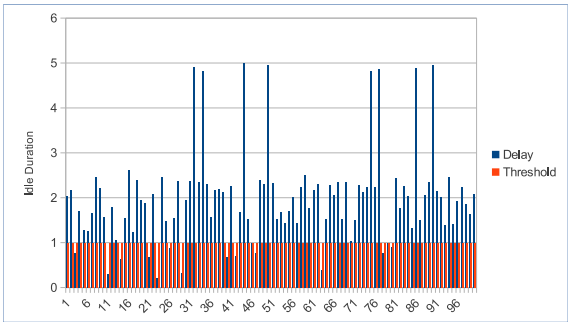

Figure 10.9: Slack degree d = 6, input set 1

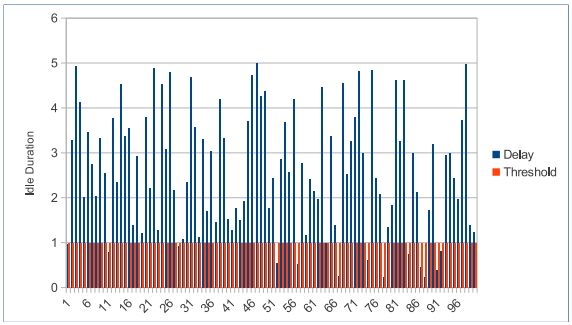

Figure 10.10: Slack degree d = 6, input set 2

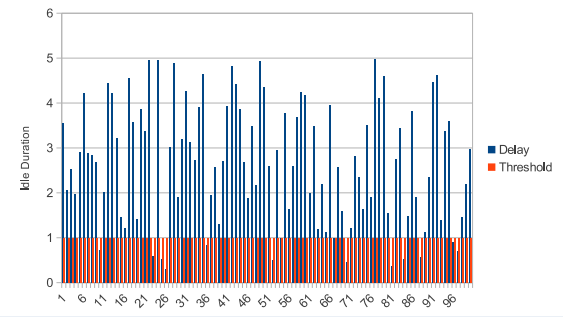

Figure 10.11: Slack degree d = 6, input set 3

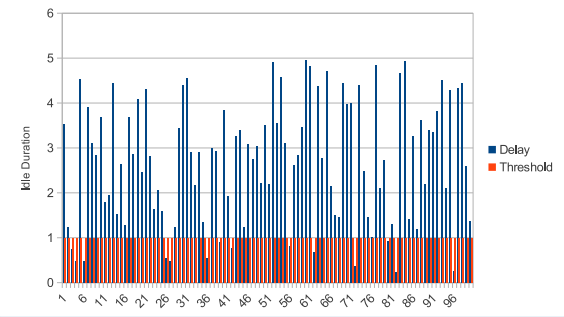

Figure 10.12: Slack degree d = 6, input set 4

DRA			Budget Based Algorithm			OWCR	OPT
$\epsilon = 0.001$	$\epsilon = 0.01$	$\epsilon = 0.1$	$\epsilon = 0.0005$	$\epsilon = 0.005$	$\epsilon = 0.05$		
154.71	141.66	123.00	108.06	105.55	105.18	180.98	94.98
157.21	142.19	122.35	117.94	105.75	103.68	180.45	94.45
164.44	143.52	123.59	117.96	105.77	102.77	180.98	94.98
161.11	142.50	122.02	107.66	104.46	104.12	180.21	94.21

Table 10.3: Costs for d = 6

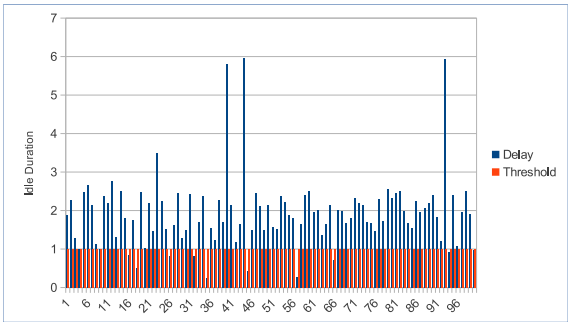

Figure 10.13: Slack degree d = 8, input set 1

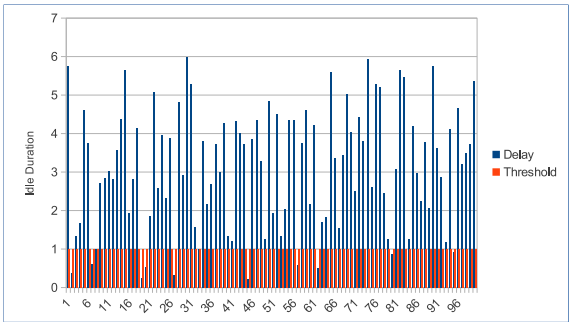

Figure 10.14: Slack degree d = 8, input set 2

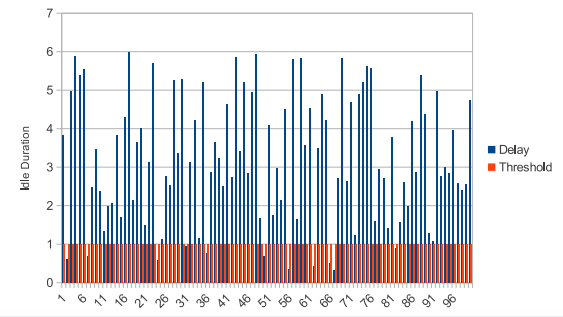

Figure 10.15: Slack degree d = 8, input set 3

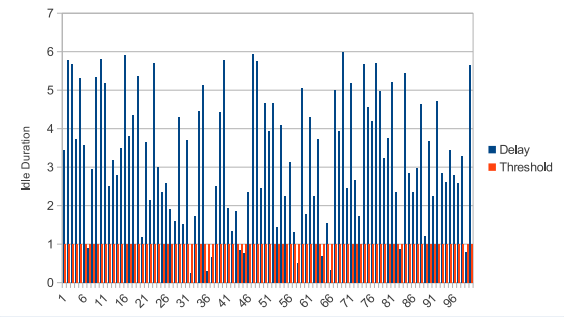

Figure 10.16: Slack degree d = 8, input set 4

DRA			Budget Based Algorithm			OWCR	OPT
$\epsilon = 0.001$	$\epsilon = 0.01$	$\epsilon = 0.1$	$\epsilon = 0.0005$	$\epsilon = 0.005$	$\epsilon = 0.05$		
165.57	145.17	119.73	116.83	105.77	102.77	185.55	96.55
162.95	144.76	120.30	107.97	104.76	102.15	184.14	95.14
157.19	142.97	120.23	108.27	105.06	102.37	184.77	95.77
152.11	136.74	116.97	108.87	107.95	102.77	184.87	95.87

Table 10.4: Costs for d = 8

	DRA			Budget Based		
	$\epsilon = 0.001$	$\epsilon = 0.01$	$\epsilon = 0.1$	$\epsilon = 0.0005$	$\epsilon = 0.005$	$\epsilon = 0.05$
Slack degree = 2	0.48%	1.21%	12.67%	28.94%	25.22%	32.30%
	0.73%	4.88%	10.80%	19.93%	27.12%	30.97%
	0.10%	2.25%	12.01%	31.10%	28.83%	28.77%
	0.11%	4.12%	14.40%	19.29%	32.12%	32.21%
Slack degree = 4	4.10%	13.90%	26.62%	36.85%	38.71%	38.94%
	5.16%	12.06%	24.29%	37.20%	39.19%	40.50%
	4.53%	13.87%	23.02%	31.26%	38.55%	40.30%
	6.53%	11.99%	23.69%	33.19%	34.52%	37.54%
Slack degree = 6	14.52%	21.73%	32.04%	40.29%	41.68%	41.88%
	12.88%	21.20%	32.30%	34.64%	41.40%	42.54%
	9.09%	20.70%	31.71%	34.82%	41.56%	43.21%
	10.60%	20.93%	32.30%	40.26%	42.03%	42.22%
Slack degree = 8	10.77%	21.76%	35.50%	37.04%	43.00%	44.61%
	11.51%	21.40%	34.67%	41.37%	43.11%	44.53%
	14.93%	22.62%	34.93%	41.40%	43.12%	44.60%
	17.72%	26.03%	36.73%	41.11%	41.61%	44.41%

Table 10.5: Comparison of DRA and Budget Based Algorithm with OWCR with slack system

Table 10.5 shows the percent savings for DRA and the budget based over OWCR, using the outputs generated from tables 10.1, 10.2, 10.3, and 10.4. We can see that as we increase the slack degree the DRA and budget based algorithm both have significant savings over the OWCR. This is due to the fact that once we have several consecutive slack requests, the tapering down techniques wait times converge to 0 and they both start mimicking the optimal offline strategy which is to power down if a slack request arrives which the tapering down techniques are doing. We see both the DRA and budget based techniques yield better results for larger ϵ values and for input sequences with higher slack degree because larger ϵ values force the tapering techniques to behave more aggressively and taper down at a larger rate and will converge to a wait time of 0 after fewer consecutive slack requests than when ϵ is smaller.

When we compare DRA with the budget based technique we see that the budget based technique yields significantly greater savings than the DRA when we have a large or small ϵ value. The DRA

does not have as significant savings for a small ϵ value compared to a larger ϵ value since it tapers down slowly for a small ϵ value and for a slack system it is more favorable to taper down quickly, the budget based technique, however, yields significant savings regardless of the value of ϵ, although it does improve for larger ϵ value as well, but the savings does not increase drastically for larger ϵ values. The next section will further analyze how the two algorithms behave using the same inputs used in this section with the same ϵ values.

10.2 Comparison of DRA with Budget Based Algorithm with Slack Systems

When we compared the DRA with the budget based algorithm, we saw that the budget based technique gave us better results as the slack degree increased for the input, even when the value of ϵ was decreased. We can take a look at the costs of DRA and budget based algorithm after each request for the slack degree 2 input sets using $\epsilon = 0.001$ for DRA and $\epsilon = 0.0005$ for the budget based approach such that they both have a competitive ratio of 2.001.

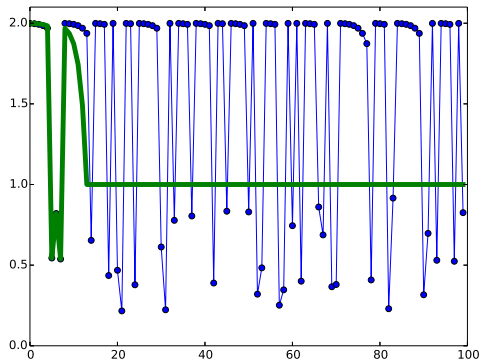

Figure 10.17: Costs when CR set to 2.001, slack degree 2, from first input set

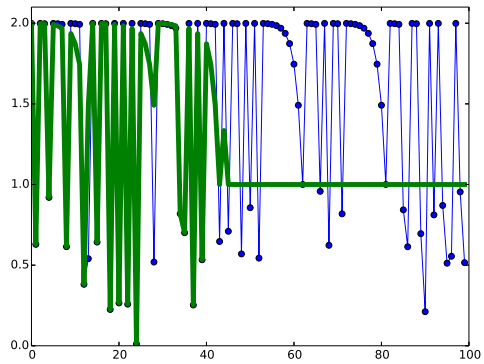

Figure 10.18: Costs when CR set to 2.001, slack degree 2, from second input set

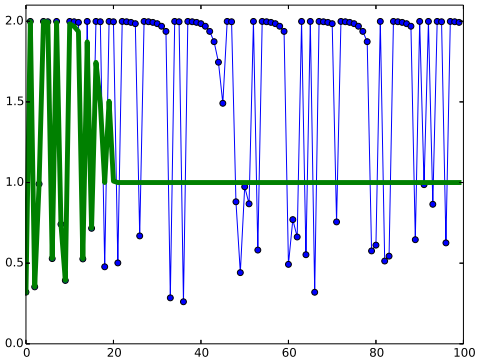

Figure 10.19: Costs when CR set to 2.001, slack degree 2, from third input set

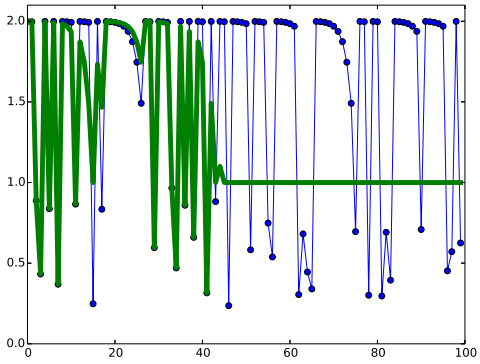

Figure 10.20: Costs when CR set to 2.001, slack degree 2, from third input set

We can see that in each figure, the budget based algorithm converges to a constant value after a set of slack requests. The DRA however constantly resets to the maximum delay time, since a busy request arrives. We see at the beginning, the cost of the budget based algorithm is similar to DRA, however once the budget based algorithm curve becomes constant the DRA is often above the budget based curve, so the budget based algorithm has a lower cost than the DRA. Let us take a look at how the two algorithms are tapering after each request.

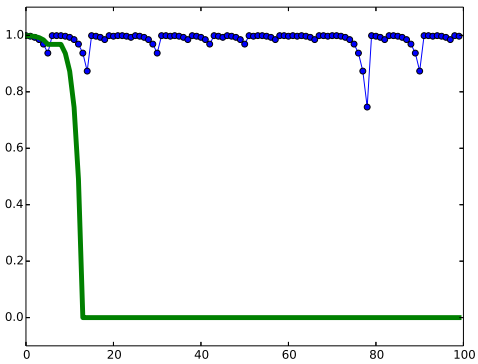

Figure 10.21: Wait times when CR set to 2.001, slack degree 2, from first input set

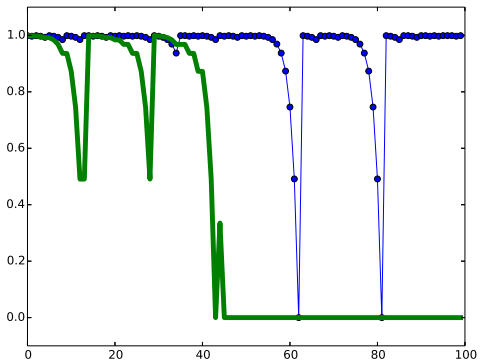

Figure 10.22: Wait times when CR set to 2.001, slack degree 2, from second input set

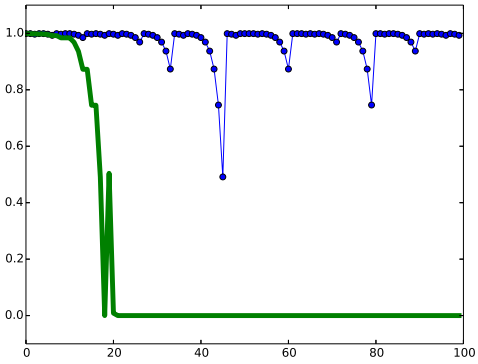

Figure 10.23: Wait times when CR set to 2.001, slack degree 2, from third input set

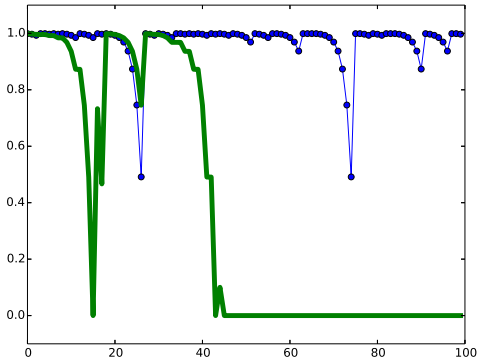

Figure 10.24: Wait times when CR set to 2.001, slack degree 2, from third input set

According to figures 10.21, 10.22, 10.23, 10.24, we see that often the wait times for the budget based technique is smaller than the wait times for the DRA. We see that DRA is tapering down at a slow rate since the value of $\epsilon = 0.001$, so it is constantly tapering down for each slack request and resets back to the initial wait time, where the budget based technique at a certain point has a wait time of 0. When a busy request arrives, the budget will decrease since the busy request incurs a loss however this loss does not decrease the budget enough to force the wait time to increase. So when the wait time is 0 for a slack request, the budget based technique is behaving similarly to the optimal offline algorithm, which yields are fairly optimal cost over the DRA. Let us increase the ϵ values such that both algorithms have a competitive ratio of 2.1.

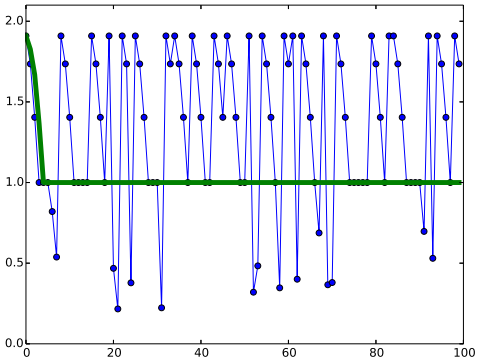

Figure 10.25: Costs when CR set to 2.1, slack degree 2, from first input set

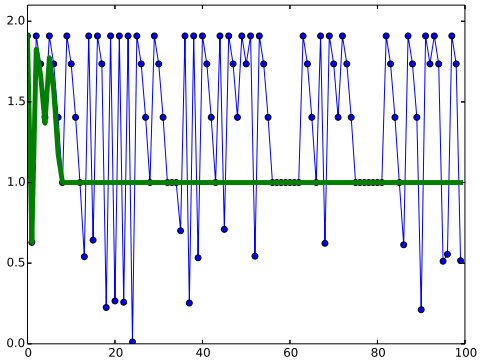

Figure 10.26: Costs when CR set to 2.1, slack degree 2, from second input set

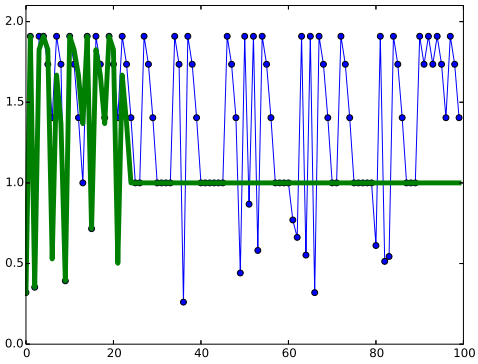

Figure 10.27: Costs when CR set to 2.1, slack degree 2, from third input set

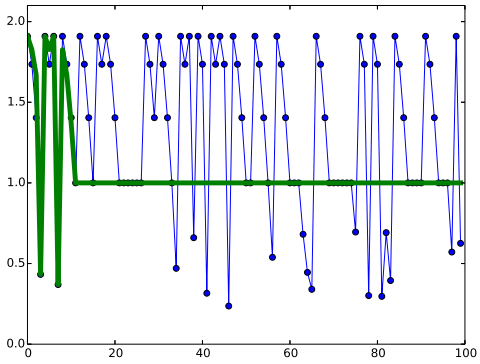

Figure 10.28: Costs when CR set to 2.1, slack degree 2, from third input set

From figures 10.25, 10.26, 10.27, and 10.28 where the competitive ratio is 2.1, we see similar behaviors as when we had smaller ϵ values which had a competitive ratio of 2.001. When the competitive ratio is 2.1, we see that the DRA has more instances when it has a minimal cost than the budget based technique however more often the budget based technique has a lower cost than the DRA for the set of requests. Let us look at how the wait times change for each request.

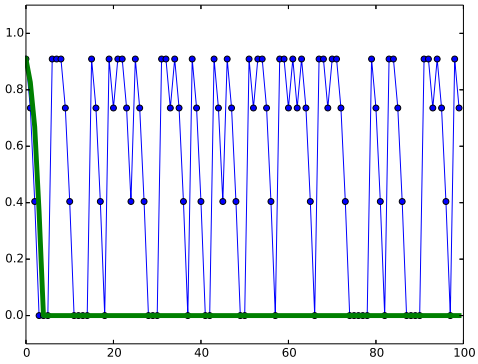

Figure 10.29: Wait times when CR set to 2.1, slack degree 2, from first input set

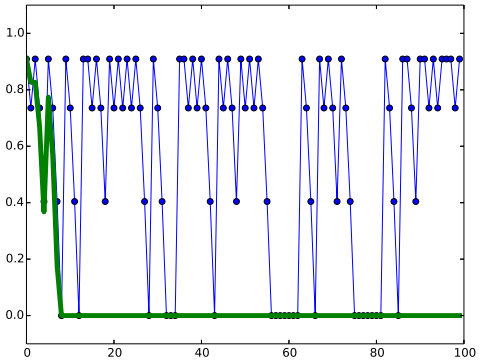

Figure 10.30: Wait times when CR set to 2.1, slack degree 2, from second input set

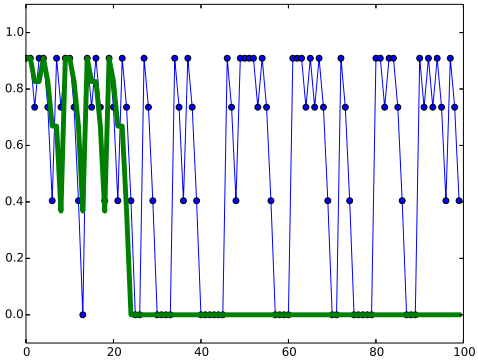

Figure 10.31: Wait times when CR set to 2.1, slack degree 2, from third input set

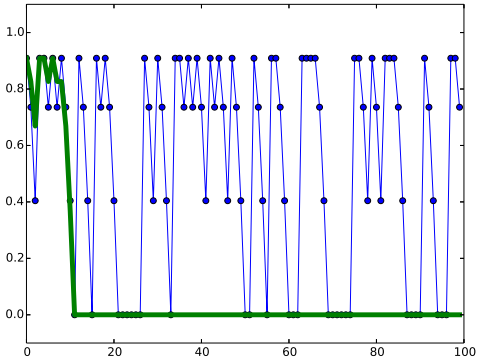

Figure 10.32: Wait times when CR set to 2.1, slack degree 2, from third input set

When we increase the competitive ratio to 2.1, we see similar behavior of the two algorithms. The wait time for the budget based reaches 0 after few number of requests, and once again, once the wait time becomes 0 it stays 0 throughout the duration of the remaining requests. The DRA tapers down at a faster rate than when the competitive ratio was 2.001, however its wait times are usually above the budget based wait time curve, there are a few exceptions where the two curves are both at 0, but overall the DRA has a larger wait time and when the cost of DRA is above the budget based, the budget based technique (as in the earlier example) is saving power and is nearly mimicking the optimal offline algorithm. Now let us look at the two algorithms for a more slack system where the slack degree is 8.

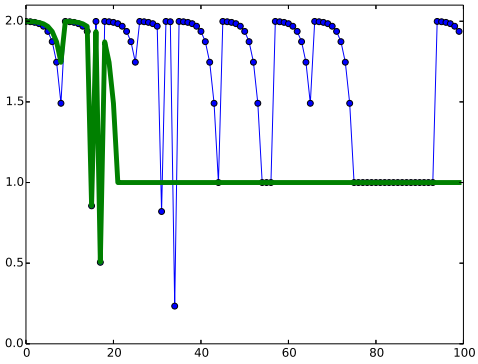

Figure 10.33: Costs when CR set to 2.001, slack degree 8, from first input set

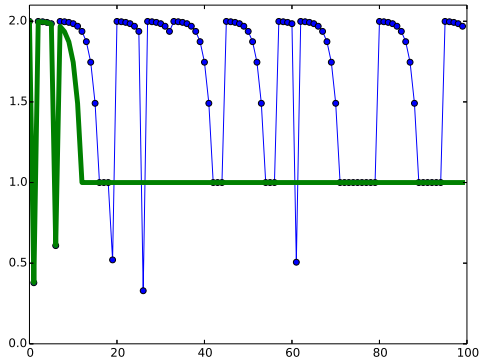

Figure 10.34: Costs when CR set to 2.001, slack degree 8, from second input set

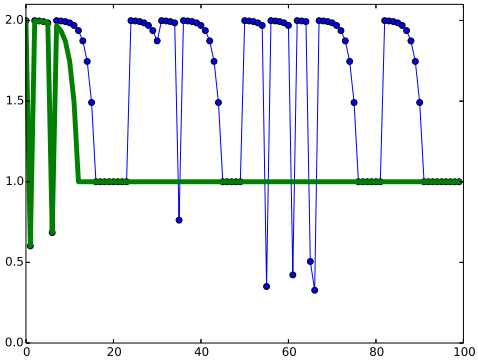

Figure 10.35: Costs when CR set to 2.001, slack degree 8, from third input set

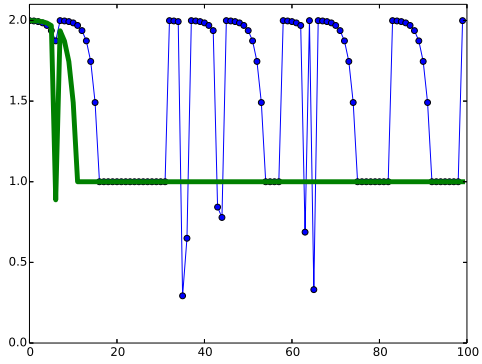

Figure 10.36: Costs when CR set to 2.001, slack degree 8, from third input set

We see similar results to the input sets where the slack degree was 2, we also see more instances where the cost of DRA is equal to the cost of the budget based approach since there are several consecutive slack requests, which is a result of having an input that is slack. The budget based approach curve once again converges to a constant after several slack degrees arrive. We can see the corresponding wait times below.

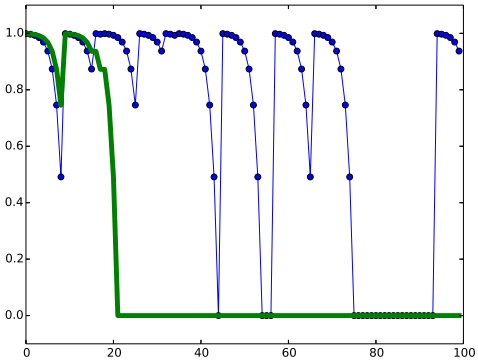

Figure 10.37: Wait times when CR set to 2.001, slack degree 8, from first input set

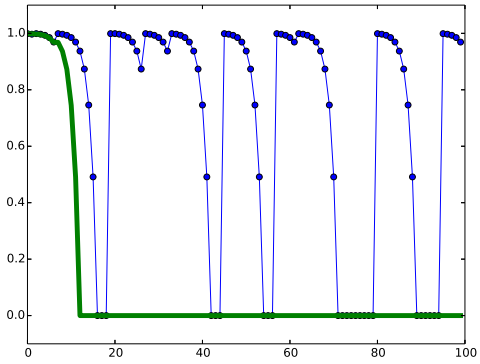

Figure 10.38: Wait times when CR set to 2.001, slack degree 8, from second input set

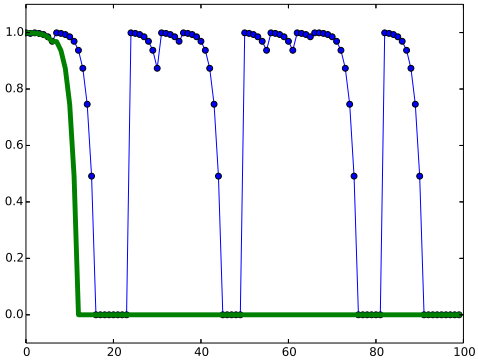

Figure 10.39: Wait times when CR set to 2.001, slack degree 8, from third input set

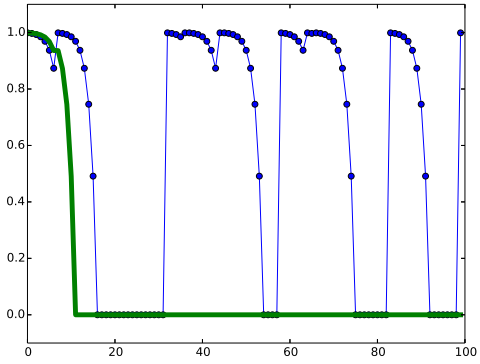

Figure 10.40: Wait times when CR set to 2.001, slack degree 8, from third input set

We can see the budget based approach tapers down to 0 after a set of slack requests arrive and the DRA is tapering down slowly since the ϵ value is arbitrarily small. There are several instances where the wait time for the DRA becomes 0 and matches the wait time of the budget based algorithm. Unlike the budget based technique, once a busy request arrives, it resets it wait time where the budget based technique remains 0 due to the fact the budget did not decrease enough to force the wait time to decrease, similarly to the example where we had a slack degree 2. Let us look at the two techniques when we increase their respective ϵ values.

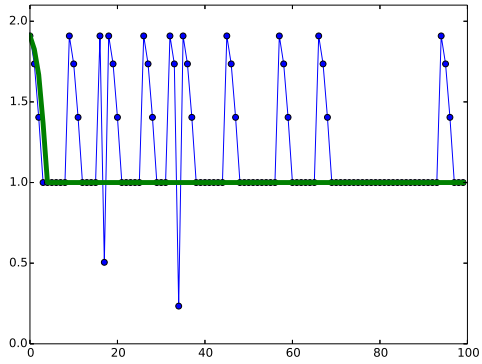

Figure 10.41: Costs when CR set to 2.1, slack degree 8, from first input set

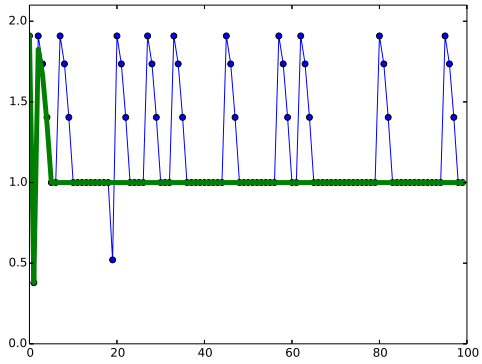

Figure 10.42: Costs when CR set to 2.1, slack degree 8, from second input set

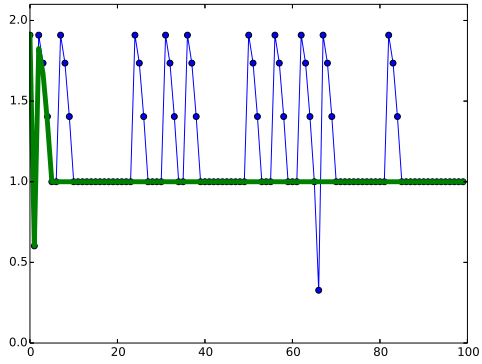

Figure 10.43: Costs when CR set to 2.1, slack degree 8, from third input set

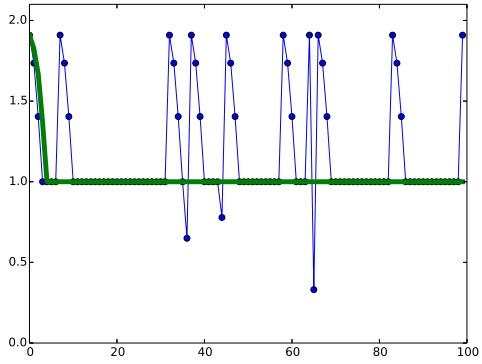

Figure 10.44: Costs when CR set to 2.1, slack degree 8, from third input set

With a high slack degree and a small ϵ value, the DRA obtained its minimal cost and we see that there are several instances where the costs of DRA amd budget based technique are equal. Since the slack degree is arbitrarily high, most of the requests in the sequence will be slack, so when the budget based cost is 1 (the power up cost), so its cost will be close the the optimal cost produced by the optimal offline algorithm which table 10.5 clearly showed. Even though the DRA was the most optimal in this case, it was wasting energy tapering down to a lower wait time when the optimal approach was to instantly power down after each slack request. The following table will show the wait times of the two algorithms.

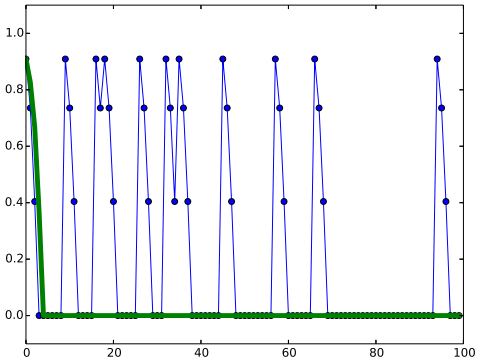

Figure 10.45: Wait times when CR set to 2.1, slack degree 8, from first input set

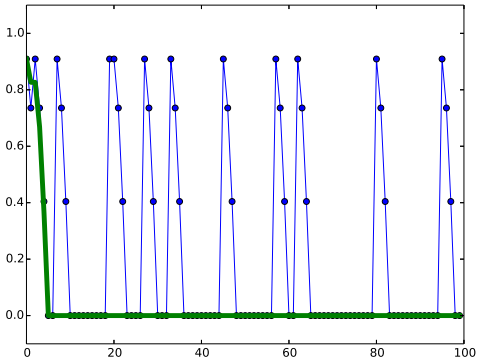

Figure 10.46: Wait times when CR set to 2.1, slack degree 8, from second input set

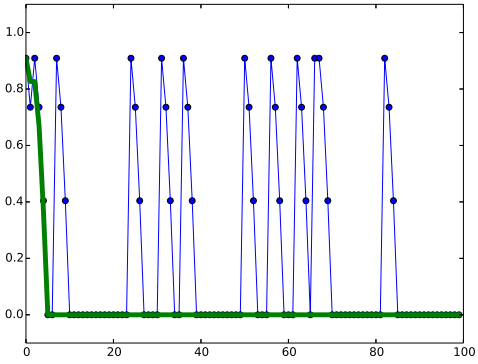

Figure 10.47: Wait times when CR set to 2.1, slack degree 8, from third input set

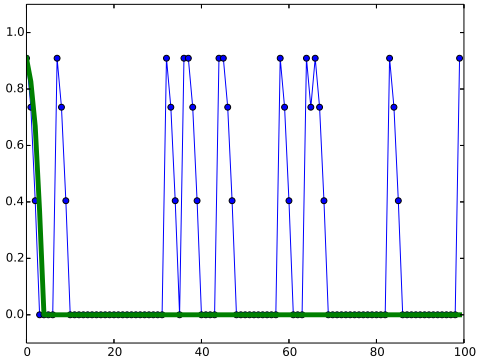

Figure 10.48: Wait times when CR set to 2.1, slack degree 8, from third input set

Based on the costs for DRA for a high slack degree and arbitrarily small ϵ value, we see that it tapers down quickly to 0 and there are many instances where the DRA and budget based algorithm both have a wait time of 0, which is the optimal wait time for any slack request, and because DRA has many instances where it is mimicking the optimal offline algorithm, DRA yields its best results compared to when we have a smaller slack degree input and when the ϵ value is smaller. We do not see a big difference for the budget based technique for slack degree 8 for when we have a smaller or larger ϵ, but DRA saw significantly better costs when ϵ was larger since it would taper down at a faster rate. In general, these tapering down algorithms improve when we have a larger slack degree and when ϵ is larger, and overall we see that for a slack system, the budget based technique is more

optimal than DRA for all the experimental inputs and for the ϵ values used. Now we will analyze the two approaches for busy systems.

10.3 Busy Systems

Earlier in this chapter, we compared DRA and the budget based technique with OWCR for inputs that were slack, where the slack degree was greater than 1. In this section we will compare the two algorithms when we have a busy input sequence, where slack degree is smaller than 1.

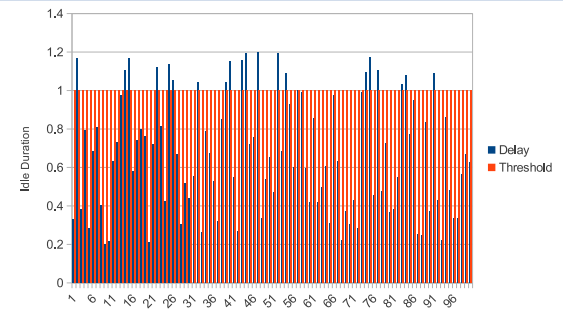

Figure 10.49: Slack degree d = 0.25, input set 1

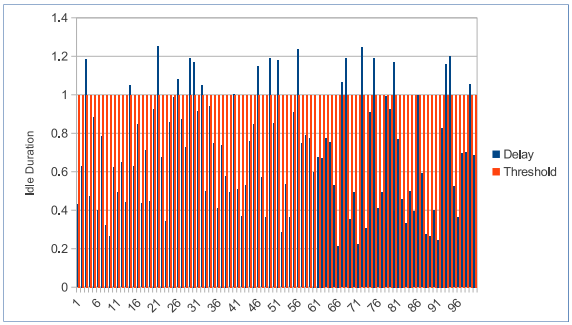

Figure 10.50: Slack degree d = 0.25, input set 2

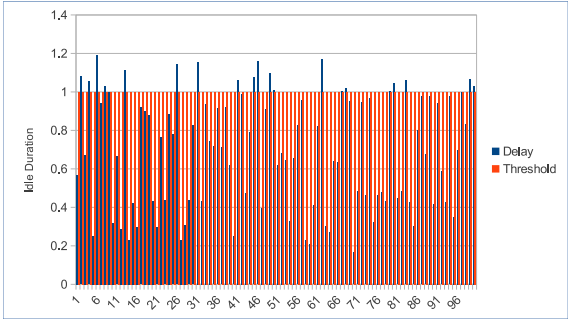

Figure 10.51: Slack degree d = 0.25, input set 3

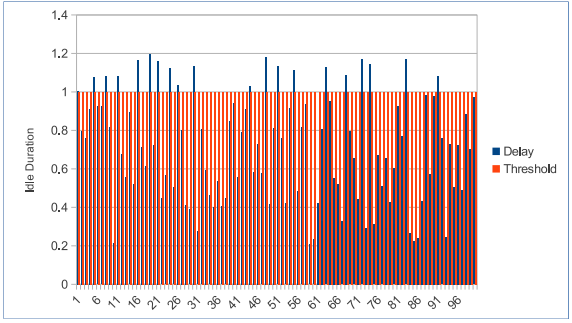

Figure 10.52: Slack degree d = 0.25, input set 4

Once again, the requests in the shaded region will be busy requests, and any request above the shaded region is slack. Once again we set $\beta = 1$ and $\alpha = 1$, so the power down threshold will be 1 idle time unit. The costs for DRA, budget based algorithm, OWCR, and OPT for each input sequence is shown in the following table.

DRA			Budget Based Algorithm			OWCR	OPT
$\epsilon = 0.001$	$\epsilon = 0.01$	$\epsilon = 0.1$	$\epsilon = 0.0005$	$\epsilon = 0.005$	$\epsilon = 0.05$		
84.64	86.34	93.81	85.98	88.96	94.08	84.67	64.67
87.67	88.43	97.12	89.05	92.78	100.83	87.70	67.70
89.75	92.41	102.04	92.51	96.04	101.39	88.78	67.78
89.38	89.21	104.55	91.49	93.01	103.76	89.40	69.40

Table 10.6: Costs for d = 0.25

So far, after looking at the input sequences in which the slack degree is 0.25, we see that DRA has a favorable cost when ϵ is smaller than when ϵ is larger, the opposite was true when the slack degree was larger than 1. We will perform the same routine for slack degree, 0.5, 2/3, and 0.8. We also notice that in this case, the budget based technique did not produce minimal costs as in the previous section, we can see that it did worse than DRA and OWCR.

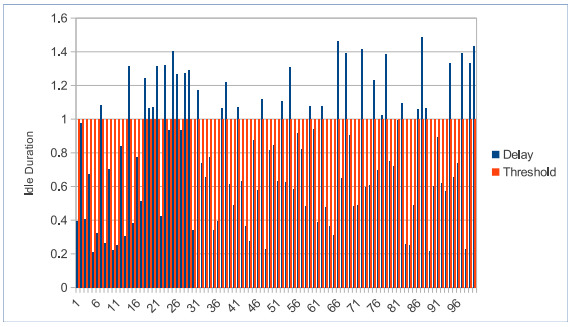

Figure 10.53: Slack degree d = 0.5, input set 1

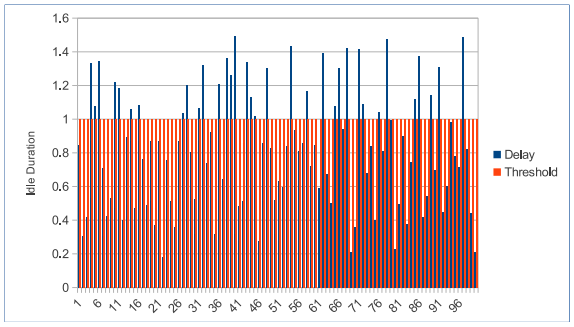

Figure 10.54: Slack degree d = 0.5, input set 2

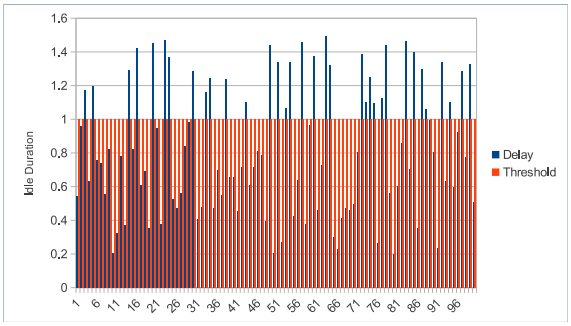

Figure 10.55: Slack degree d = 0.5, input set 3

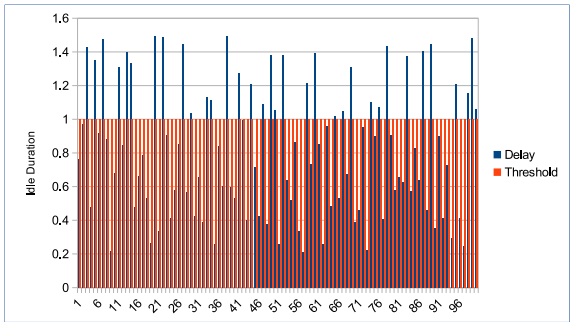

Figure 10.56: Slack degree d = 0.5, input set 4

DRA			Budget Based Algorithm			OWCR	OPT
$\epsilon = 0.001$	$\epsilon = 0.01$	$\epsilon = 0.1$	$\epsilon = 0.0005$	$\epsilon = 0.005$	$\epsilon = 0.05$		
105.42	106.76	108.79	108.49	109.94	97.84	105.50	71.50
109.08	110.42	115.12	111.30	111.70	115.65	109.15	75.15
107.44	106.78	113.01	108.82	106.70	114.58	106.51	72.51
106.57	107.11	112.77	108.93	107.85	116.10	106.61	72.61

Table 10.7: Costs for d = 0.5

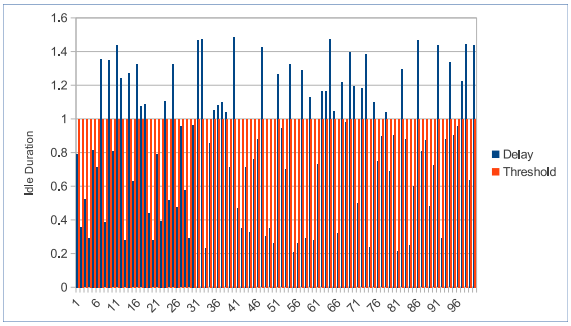

Figure 10.57: Slack degree d = 2/3, input set 1

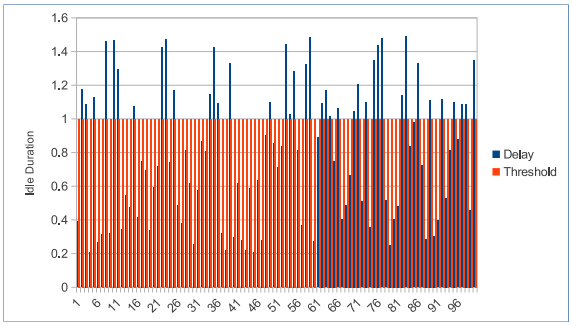

Figure 10.58: Slack degree d = 2/3, input set 2

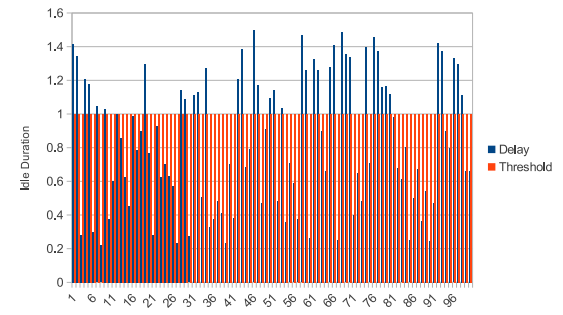

Figure 10.59: Slack degree d = 2/3, input set 3

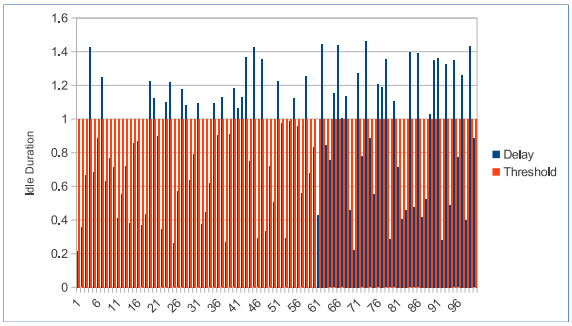

Figure 10.60: Slack degree d = 2/3, input set 4

DRA			Budget Based Algorithm			OWCR	OPT
$\epsilon = 0.001$	$\epsilon = 0.01$	$\epsilon = 0.1$	$\epsilon = 0.0005$	$\epsilon = 0.005$	$\epsilon = 0.05$		
114.57	115.64	118.44	108.17	109.30	111.32	114.68	74.68
110.17	109.42	110.12	105.12	113.86	116.00	110.26	71.26
114.48	112.78	114.43	115.80	107.67	109.24	113.66	73.66
115.40	116.36	118.07	116.95	109.15	107.05	115.47	75.47

Table 10.8: Costs for d = 2/3

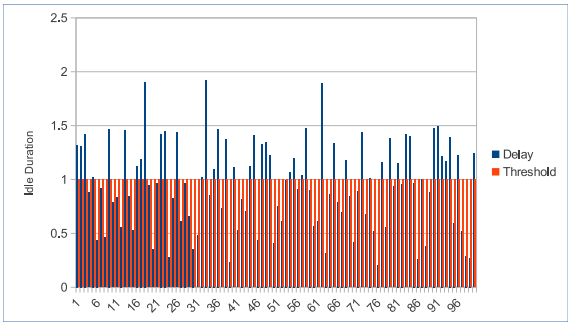

Figure 10.61: Slack degree d = 0.8, input set 1

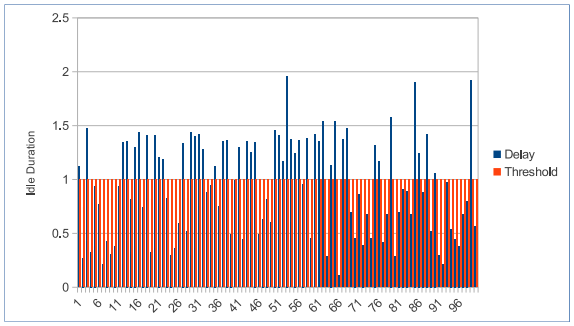

Figure 10.62: Slack degree d = 0.8, input set 2

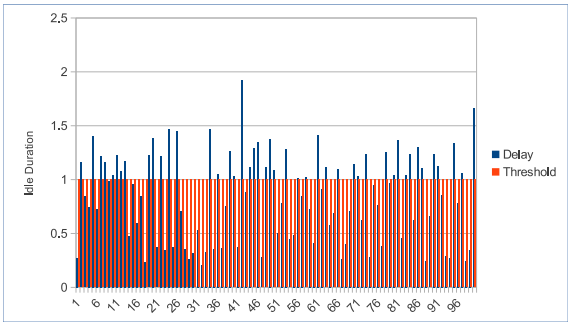

Figure 10.63: Slack degree d = 0.8, input set 3

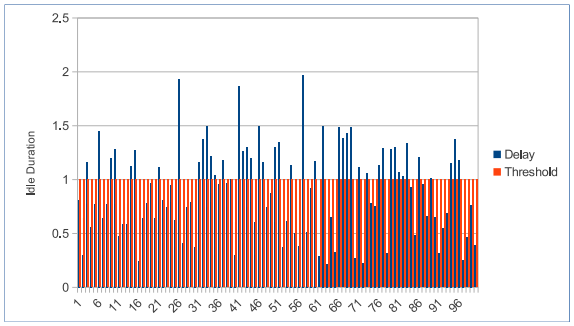

Figure 10.64: Slack degree d = 0.8, input set 4

DRA			Budget Based Algorithm			OWCR	OPT
$\epsilon = 0.001$	$\epsilon = 0.01$	$\epsilon = 0.1$	$\epsilon = 0.0005$	$\epsilon = 0.005$	$\epsilon = 0.05$		
125.45	128.90	129.59	128.40	131.46	117.75	125.58	80.58
122.73	121.01	121.72	116.60	103.86	108.15	122.29	77.29
119.89	119.96	123.87	110.75	124.22	110.07	119.99	74.99
123.70	121.75	119.03	118.56	104.13	104.72	122.93	77.93

Table 10.9: Costs for d = 0.8

	DRA			Budget Based		
	$\epsilon = 0.001$	$\epsilon = 0.01$	$\epsilon = 0.1$	$\epsilon = 0.0005$	$\epsilon = 0.005$	$\epsilon = 0.05$
Slack degree = 0.25	0.04%	-1.97%	-10.80%	-1.55%	-5.07%	-11.11%
	0.03%	-0.80%	-10.74%	-1.54%	-5.80%	-15.00%
	-1.13%	-4.10%	-14.94%	-4.20%	-8.18%	-14.20%
	0.02%	0.21%	-16.95%	-2.34%	-4.04%	-16.06%
Slack degree = 0.5	0.08%	-1.19%	-3.12%	-2.83%	-4.21%	7.26%
	0.06%	-1.16%	-5.50%	-2.00%	-2.34%	-5.96%
	-0.87%	-0.25%	-6.10%	-2.17%	-0.18%	-7.58%
	0.04%	-0.47%	-5.78%	-2.18%	-1.16%	-8.90%
Slack degree = 2/3	0.10%	-0.84%	-3.01%	5.68%	4.69%	2.92%
	0.08%	0.76%	0.13%	4.66%	-3.27%	-5.21%
	-0.72%	0.77%	-0.68%	-1.88%	5.27%	3.89%
	0.06%	-0.77%	-2.25%	-1.28%	5.47%	7.29%
Slack degree = 0.8	0.10%	-2.64%	-3.19%	-2.25%	-4.68%	6.24%
	-0.36%	1.05%	0.47%	4.65%	15.07%	11.56%
	0.08%	0.03%	-3.23%	7.70%	-3.53%	8.27%
	-0.65%	0.96%	3.17%	3.55%	15.29%	14.81%

Table 10.10: Comparison of DRA and Budget Based Algorithm with OWCR with busy system

Table 10.10 shows the percent savings for DRA and the budget based over OWCR using outputs from tables 10.6, 10.7, 10.8, and 10.9. For busier systems. We can see that the tapering techniques do not yield a significant savings, we actually use more energy when the slack degree is arbitrarily

smaller and when the ϵ value is larger. The reason behind this is because if the tapering techniques taper down too quickly when a slack request does arrive, tapering down will cause them to lose energy since it is likely that we will have a busy request and the optimal strategy is not to power down for said strategy. So we see for a arbitrarily busier system, both techniques obtain better costs when when ϵ is smaller.

If we compare DRA with the budget based technique, we see that when the slack degree is smaller, the DRA usually has better performance than the budget based technique. This is due to the fact that DRA tapers down at a smaller rate and resets back more rapidly than the budget based technique and for a busy system, having the wait time close to the threshold wait time is in fact the optima strategy, we will see in the next section that shows this behavior. We can also see that when the slack degree increases, the budget based technique obtains significantly better results than DRA. The final conclusion that can be seen is that if we have an arbitrarily busy system, OWCR is the better strategy than a tapering down strategy.

10.4 Comparison of DRA with Budget Based Algorithm with Busy Systems

Let us analyze the costs of DRA with budget based after each input sequence used in the previous section.

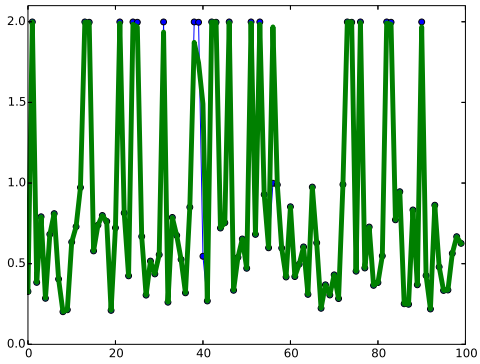

Figure 10.65: Costs when CR set to 2.001, slack degree 0.25, from first input set

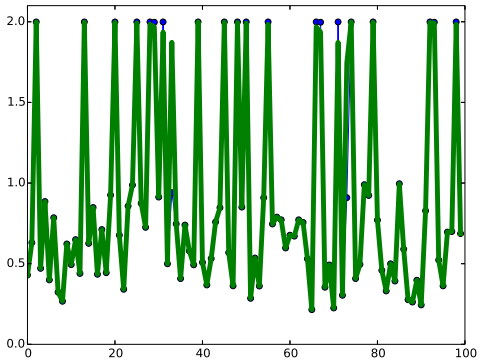

Figure 10.66: Costs when CR set to 2.001, slack degree 0.25, from second input set

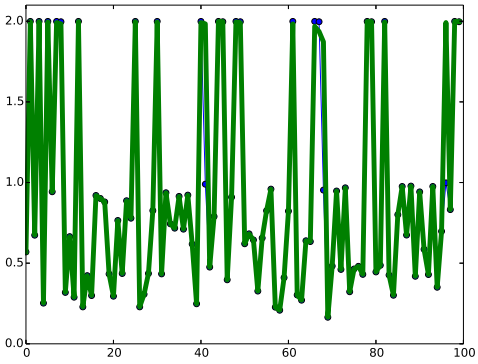

Figure 10.67: Costs when CR set to 2.001, slack degree 0.25, from third input set

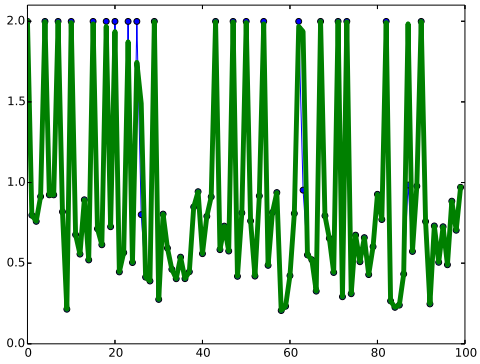

Figure 10.68: Costs when CR set to 2.001, slack degree 0.25, from forth input set

When ϵ for DRA and budget based was 0.001 and 0.0005 respectively, the taper down techniques were the most optimal for a busy systems, where most of the requests were busy requests. Their costs were similar to OWCR however DRA was slightly more optimal than the budget based technique. However even though DRA was slightly more optimal, the cost curves looked identical after most of the requests. Let us take a look at the tapered down wait times.

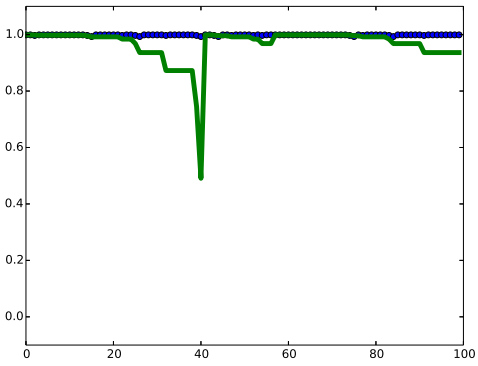

Figure 10.69: Wait times when CR set to 2.001, slack degree 0.25, from first input set

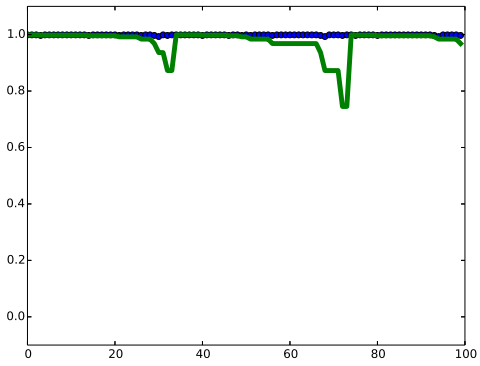

Figure 10.70: Wait times when CR set to 2.001, slack degree 0.25, from second input set

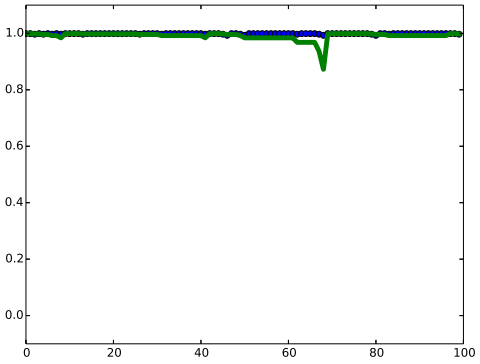

Figure 10.71: Wait times when CR set to 2.001, slack degree 0.25, from third input set

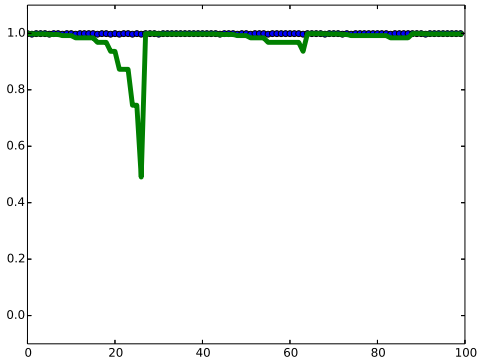

Figure 10.72: Wait times when CR set to 2.001, slack degree 0.25, from forth input set

We can see that the wait times for DRA are all converged to 1 (since β and α are both 1, the maximum wait time will be 1 as well), so DRA is basically mimicking the OWCR which happens to be the optimal approach when the slack degree is arbitrarily small, when we have a busy system. The budget based technique has a instance where the wait time drops and then tapers back up to 1. During these intervals is where DRA yields power savings since the machine running the budget based technique is powering down too soon and the request happens to be a busy request and the DRA saved power by remaining in the on state at the moment when that busy request arrives. We can see that pattern for of the test inputs. Now let us see the cost curves when we increase the ϵ values.

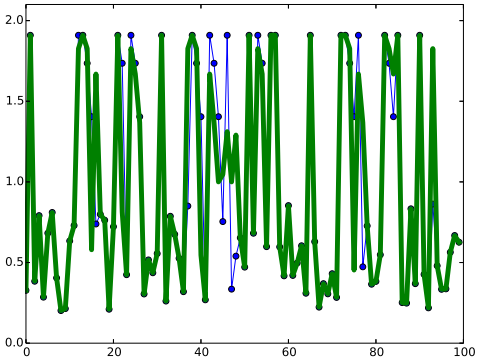

Figure 10.73: Costs when CR set to 2.1, slack degree 0.25, from first input set

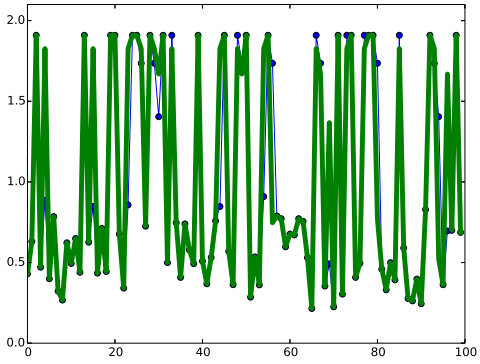

Figure 10.74: Costs when CR set to 2.1, slack degree 0.25, from second input set

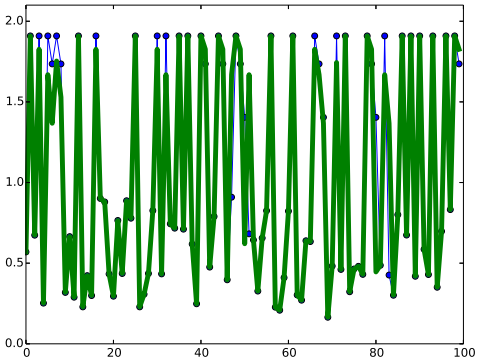

Figure 10.75: Costs when CR set to 2.1, slack degree 0.25, from third input set

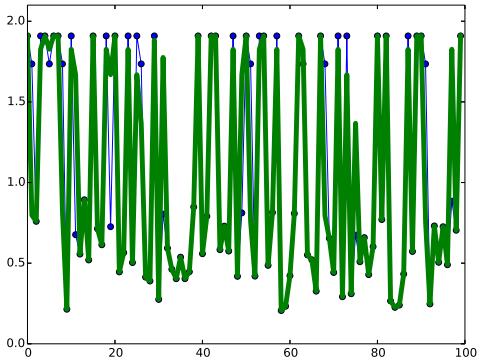

Figure 10.76: Costs when CR set to 2.1, slack degree 0.25, from forth input set

We see when the value of ϵ is larger, the cost curves of the two algorithms differ for many of the requests. From table 10.10, we know that DRA and budget based yield worse results for slack degree 0.25 when ϵ is arbitrarily larger, which is the opposite when we have a more slack system. But once again, the DRA is slightly more optimal than the budget based approach. Once again, let us look at the tapered wait times after each request.

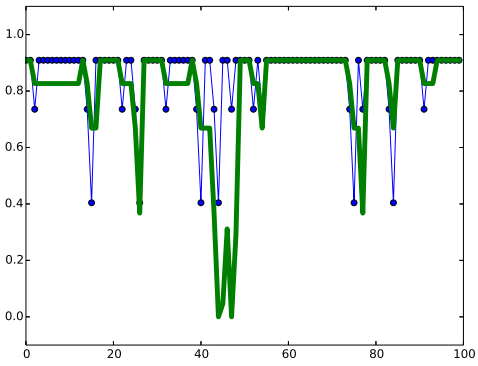

Figure 10.77: Wait times when CR set to 2.1, slack degree 0.25, from first input set

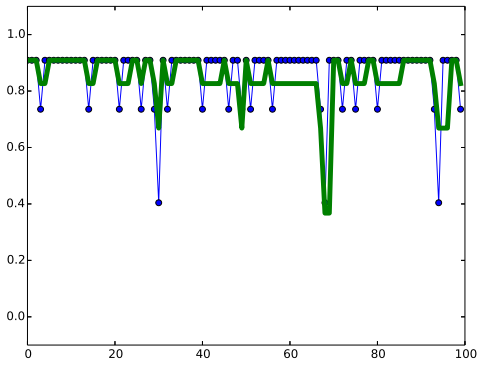

Figure 10.78: Wait times when CR set to 2.1, slack degree 0.25, from second input set

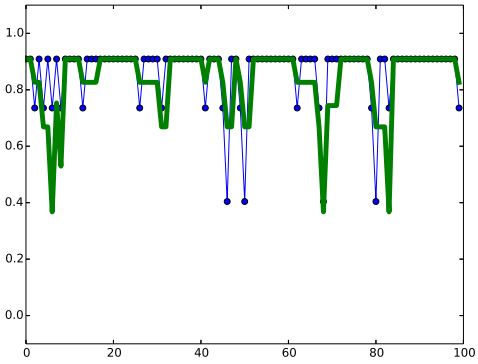

Figure 10.79: Wait times when CR set to 2.1, slack degree 0.25, from third input set

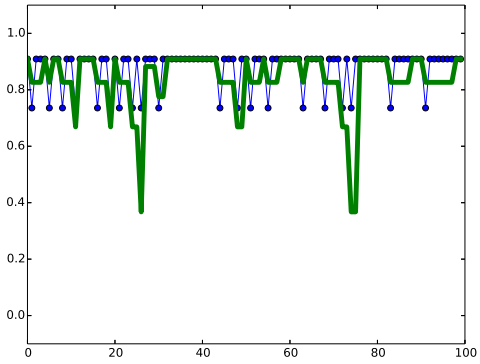

Figure 10.80: Wait times when CR set to 2.1, slack degree 0.25, from forth input set

Similarly to when ϵ was smaller, the budget based technique is tapering down at a faster rate which can cause it to save more power than DRA but because it powers down too soon when we encounter a busy request which causes the budget to decrease and causing the budget based technique to yield a larger cost than the DRA approach. Both techniques are significantly worse than OWCR, when ϵ is larger, due to the fact that they taper down too quickly and incur a power up cost for several busy requests in which power could have been saved had the machine not powered down. We can see that when we had a slack system, the budget based technique was the better algorithm but for a busy system, we see that DRA is more optimal but OWCR is more optimal than both techniques for busier system. Let us take a look at the costs when the input requests are more slack, we will analyze when the slack degree is 0.8.

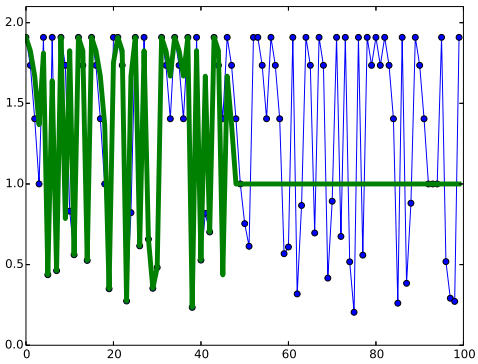

Figure 10.81: Costs when CR set to 2.1, slack degree 0.8, from first input set

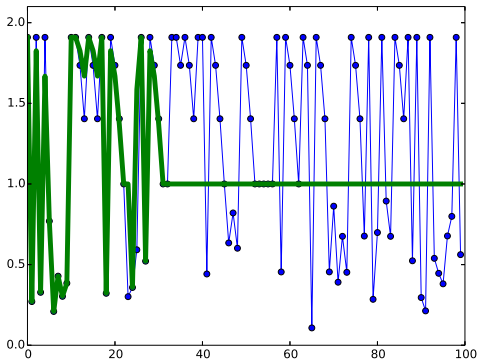

Figure 10.82: Costs when CR set to 2.1, slack degree 0.8, from second input set

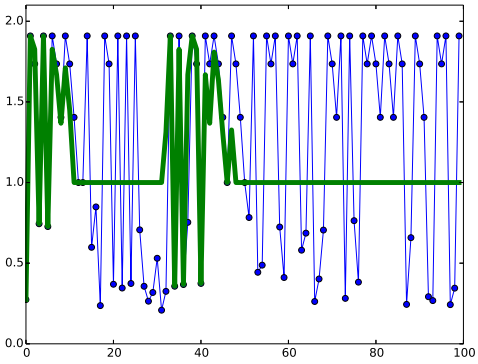

Figure 10.83: Costs when CR set to 2.1, slack degree 0.8, from third input set

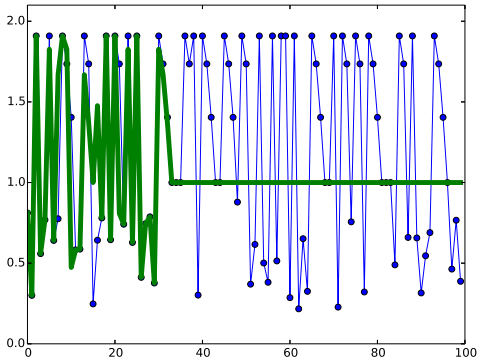

Figure 10.84: Costs when CR set to 2.1, slack degree 0.8, from forth input set

In the above figures we are seeing similar results as to when the slack degree was greater than 1. Once again the budget based technique is saving energy due to the fact that there are more slack requests and the budget increases. The budget never decreases enough to force the wait time for the budget based technique to increase to a value above 0. Now we see that the DRA curve has many instances where its cost is greater than 1 during the times when the budget based technique is converged at 1. This explains the savings for the budget based technique over the DRA when the slack degree is set to 0.8.

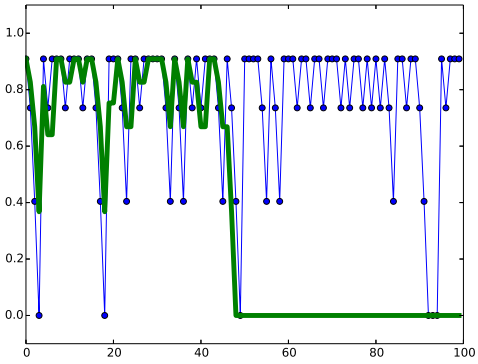

Figure 10.85: Wait times when CR set to 2.1, slack degree 0.8, from first input set

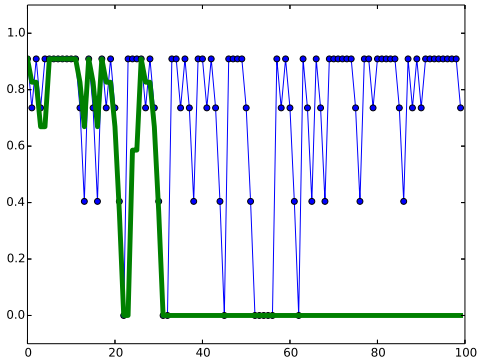

Figure 10.86: Wait times when CR set to 2.1, slack degree 0.8, from second input set

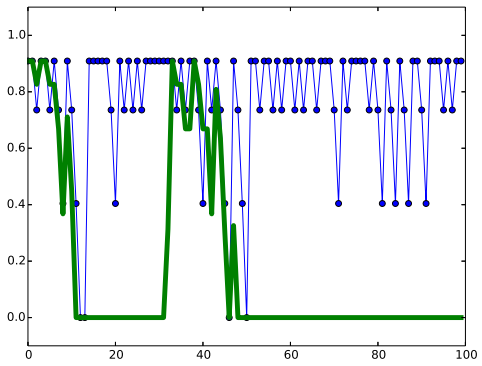

Figure 10.87: Wait times when CR set to 2.1, slack degree 0.8, from third input set

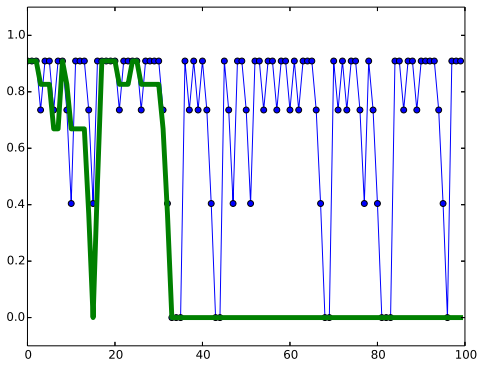

Figure 10.88: Wait times when CR set to 2.1, slack degree 0.8, from forth input set

Once again, the budget based wait times become 0, and the DRA is slowly tapering down to 0 and then it resets once we have a busy request, but the budget based wait time remains at 0 since the budget does not decrease significantly enough to force the wait time to increase. As we saw for a slack system, during this interval do we see that the budget based technique yields a better cost over DRA. Let us look at the situation where we decrease the ϵ values.

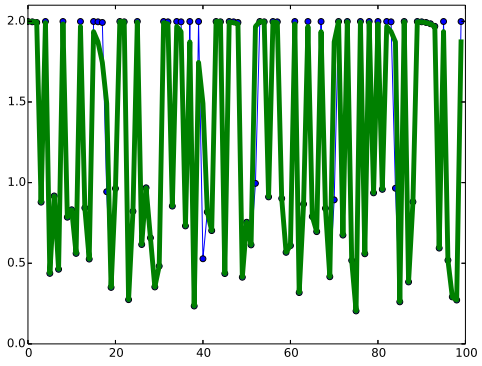

Figure 10.89: Costs when CR set to 2.001, slack degree 0.8, from first input set

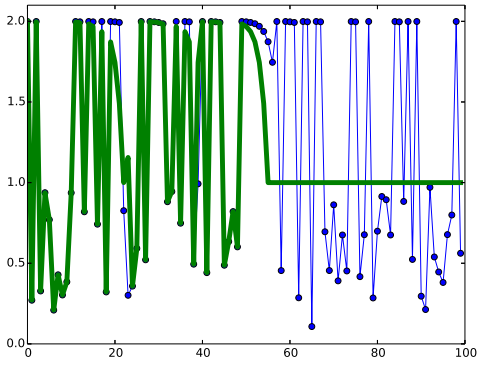

Figure 10.90: Costs when CR set to 2.001, slack degree 0.8, from second input set

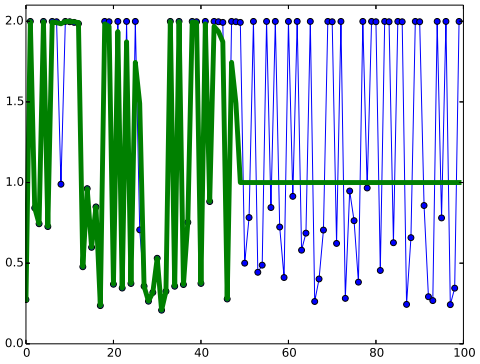

Figure 10.91: Costs when CR set to 2.001, slack degree 0.8, from third input set

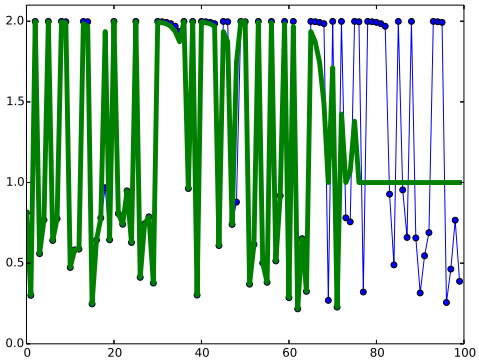

Figure 10.92: Costs when CR set to 2.001, slack degree 0.8, from forth input set

The budget based technique is more optimal than DRA even when we decrease the ϵ value however the savings is not as significant as when the ϵ value was larger. Since ϵ is smaller, the two algorithms are not tapering down as quickly, for the budget based technique, it took more time for the cost curve to converge to 1, because it took more requests to build up the budget to the point where it would force the wait time to become 0 due to the fact that the machine was not tapering as quickly.

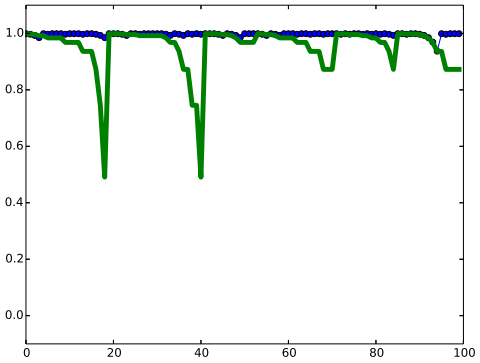

Figure 10.93: Wait times when CR set to 2.001, slack degree 0.8, from first input set

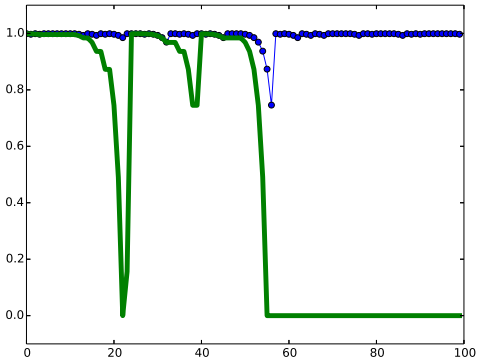

Figure 10.94: Wait times when CR set to 2.001, slack degree 0.8, from second input set

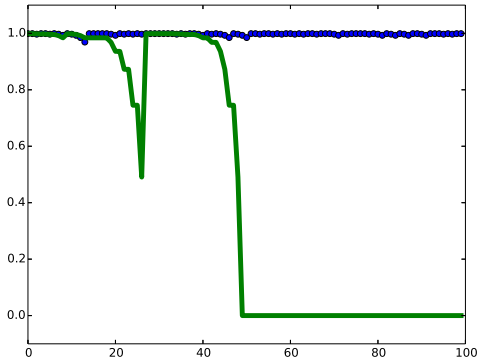

Figure 10.95: Wait times when CR set to 2.001, slack degree 0.8, from third input set

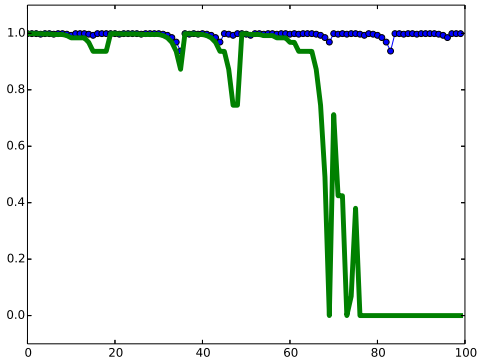

Figure 10.96: Wait times when CR set to 2.001, slack degree 0.8, from forth input set

We see that similarly to the previous example, when ϵ was larger, the wait time of the budget based technique eventually converges to 0 which yields a more optimal result than DRA since there are several slack requests and wait time of the DRA is larger than budget based wait times, which is not the optimal choice for a slack requests. Since ϵ is smaller, for one case we saw that the budget based never converged and for the other cases it converges but it took more time which is why the budget based technique for this case was not as optimal.

In conclusion, we see that when we have a more busy system, neither the budget based technique nor the DRA yielded much savings over the OWCR. For a more busy system, a smaller ϵ would yield better results and for a busy system a larger ϵ value yielded better results. For a busy system, DRA had better results than the budget based and for a more slack system, the budget based yielded better results. So in general for a busy system where the slack degree is closer to 0, the OWCR is a better choice and when the slack degree gets larger the two tapering algorithms were both better than OWCR however the budget based yielded better results than the DRA.

Chapter 11

Comparison of 3 State Taper Down Algorithms

11.1 Slack Systems

In this chapter, we will compare the three state DRA with the three state algorithm with reduced delay times using a budget which was introduced in chapter 9, for the rest of this chapter we will call this algorithm Budget Based Algorithm (BBA). The input sets will be the same as used in chapter 10, which were displayed in the figures in that chapter. We will use ϵ_{BBA} and ϵ_{DRA} to denote the ϵ to adjust the competitive ratio for BBA and DRA respectively. We choose ϵ_{BBA} and ϵ_{DRA} such that the adjusted increased competitive ratio of BBA and DRA are equal. As in chapter 10, we will run the algorithms with input in which its slack degree is greater than 1 and afterwards input sets that have a slack degree less than 1 which denote busier systems. We will also compute the cost of LEA and $\text{ALG}_{1.8}$ which denotes the algorithm which yields a competitive ratio of 1.8 when $a = 0.6$ and $d = 0.4$, the optimal algorithm for three state problems, and LEA will be the 2-competitive algorithm.

1^{st} input set	ϵ_{BBA}	BBA	ϵ_{DRA}	DRA	LEA	$\text{Alg}_{1.8}$	OPT
	1/18	101.99	0.1	123.53			
	1/180	107.34	0.01	138.63	151.81	136.93	84.81
	1/1800	105.02	0.001	143.00			
2^{nd} input set	ϵ_{BBA}	BBA	ϵ_{DRA}	DRA	LEA	$\text{Alg}_{1.8}$	OPT
	1/18	101.63	0.1	128.32			
	1/180	111.38	0.01	138.78	150.55	135.54	85.55
	1/1800	105.22	0.001	143.27			
3^{rd} input set	ϵ_{BBA}	BBA	ϵ_{DRA}	DRA	LEA	$\text{Alg}_{1.8}$	OPT
	1/18	102.36	0.1	128.12			
	1/180	110.53	0.01	141.31	153.93	138.70	86.93
	1/1800	103.79	0.001	147.39			
4^{th} input set	ϵ_{BBA}	BBA	ϵ_{DRA}	DRA	LEA	$\text{Alg}_{1.8}$	OPT
	1/18	105.29	0.1	124.96			
	1/180	103.49	0.01	138.07	153.66	138.24	86.66
	1/1800	105.08	0.001	143.87			

Table 11.1: Costs for 3 state algorithms, a = 0.6 and d = 0.4, slack degree 2

We can see from the above table that for a slack system, in this case slack degree 2, LEA and $\text{ALG}_{1.8}$ are significantly worse than DRA and BBA. For all ϵ_{BBA} values, BBA does not have any significant changes, and the costs are all similar to the cost of OPT. As we know, if ϵDRA is smaller then its cost goes up, as we have seen in the previous chapter, but BBA is not effected the same way. Which means if we decrease the ϵ_{BBA} we have a smaller competitive ratio in the worst case without increasing the overall cost for this random input set.

1^{st} input set	ϵ_{BBA}	BBA	ϵ_{DRA}	DRA	LEA	$\text{Alg}_{1.8}$	OPT
	1/18	103.80	0.1	119.80			
	1/180	103.73	0.01	135.33	172.27	154.83	92.27
	1/1800	105.98	0.001	149.85			
2^{nd} input set	ϵ_{BBA}	BBA	ϵ_{DRA}	DRA	LEA	$\text{Alg}_{1.8}$	OPT
	1/18	101.99	0.1	122.46			
	1/180	106.01	0.01	138.84	172.71	155.29	92.71
	1/1800	105.47	0.001	149.40			
3^{rd} input set	ϵ_{BBA}	BBA	ϵ_{DRA}	DRA	LEA	$\text{Alg}_{1.8}$	OPT
	1/18	101.68	0.1	119.68			
	1/180	103.71	0.01	136.27	171.57	154.54	91.57
	1/1800	105.96	0.001	149.70			
4^{th} input set	ϵ_{BBA}	BBA	ϵ_{DRA}	DRA	LEA	$\text{Alg}_{1.8}$	OPT
	1/18	105.34	0.1	124.60			
	1/180	110.09	0.01	141.64	173.03	155.68	93.03
	1/1800	111.64	0.001	151.17			

Table 11.2: Costs for 3 state algorithms, a = 0.6 and d = 0.4, slack degree 4

We can clearly see that if the input has a higher slack degree, LEA and $\text{ALG}_{1.8}$ increase even further than when the slack degree was set at 2. So for a higher slack degree, LEA, which is 2-competitive, is truly approaching a cost twice as much as OPT, same issue for $\text{ALG}_{1.8}$. Not much significant difference is seen with BBA and DRA in this input test. We once again that for DRA if ϵ_{DRA} is smaller it has a worse performance than when ϵ_{DRA} is larger, as with 2 state problem, if the system is slack, OPT will typically go into the OFF state right after the request and DRA behaves similarly for slack input and when ϵ_{DRA} is larger.

1^{st} input set	ϵ_{BBA}	BBA	ϵ_{DRA}	DRA	LEA	$\text{Alg}_{1.8}$	OPT
	1/18	103.80	0.1	117.94			
	1/180	103.71	0.01	130.44	180.98	162.67	94.98
	1/1800	106.15	0.001	141.43			
2^{nd} input set	ϵ_{BBA}	BBA	ϵ_{DRA}	DRA	LEA	$\text{Alg}_{1.8}$	OPT
	1/18	102.71	0.1	116.74			
	1/180	103.90	0.01	129.93	180.45	162.29	94.45
	1/1800	106.15	0.001	142.52			
3^{rd} input set	ϵ_{BBA}	BBA	ϵ_{DRA}	DRA	LEA	$\text{Alg}_{1.8}$	OPT
	1/18	101.99	0.1	117.01			
	1/180	104.00	0.01	131.98	180.98	162.84	94.98
	1/1800	106.22	0.001	146.59			
4^{th} input set	ϵ_{BBA}	BBA	ϵ_{DRA}	DRA	LEA	$\text{Alg}_{1.8}$	OPT
	1/18	104.21	0.1	116.13			
	1/180	104.93	0.01	130.52	180.21	162.25	94.21
	1/1800	104.88	0.001	144.19			

Table 11.3: Costs for 3 state algorithms, a = 0.6 and d = 0.4, slack degree 6

When we have slack degree 6, for LEA and $\text{ALG}_{1.8}$ has similar results as with earlier trials. BBA stills seems to be the better choice taper down approach for slack systems than the 3 state DRA, regardless of their respective ϵ values. As with the 2 state machine, even though DRA and BBA have a higher competitive ratio, they have better performance for a random input sequence. Clearly this is the case because for LEA and $\text{ALG}_{1.8}$ algorithms have a worst case cost when we have a slack request since it is in the ON or INT state more often than DRA and BBA. We will look at one last input set next.

1^{st} input set	ϵ_{BBA}	BBA	ϵ_{DRA}	DRA	LEA	$\text{Alg}_{1.8}$	OPT
	1/18	101.99	0.1	114.35			
	1/180	104.00	0.01	131.26	185.55	166.71	96.55
	1/1800	106.61	0.001	148.13			
2^{nd} input set	ϵ_{BBA}	BBA	ϵ_{DRA}	DRA	LEA	$\text{Alg}_{1.8}$	OPT
	1/18	102.48	0.1	114.83			
	1/180	105.66	0.01	132.29	184.14	165.75	95.14
	1/1800	105.22	0.001	146.50			
3^{rd} input set	ϵ_{BBA}	BBA	ϵ_{DRA}	DRA	LEA	$\text{Alg}_{1.8}$	OPT
	1/18	101.58	0.1	114.40			
	1/180	104.12	0.01	131.78	184.76	166.30	95.77
	1/1800	105.44	0.001	142.82			
4^{th} input set	ϵ_{BBA}	BBA	ϵ_{DRA}	DRA	LEA	$\text{Alg}_{1.8}$	OPT
	1/18	101.99	0.1	112.75			
	1/180	104.04	0.01	125.70	184.87	166.27	95.87
	1/1800	111.62	0.001	138.81			

Table 11.4: Costs for 3 state algorithms, a = 0.6 and d = 0.4, slack degree 8

After seeing all the slack input sequences we can conclude that for 3 state slack system, a tapering down approach yields better results. DRA has better performance when the slack degree is raised and when ϵ is larger. BBA for all input sequences never does not seem to improve but have similar results for all input sequences that we have used. Since every input sequence has a slack degree greater than 1, is always gaining thus the budget is always increasing. Once the budget is large enough, the values for y_1 and y_2 reach the value of 0. When a busy request arrives the loss is not great enough to decrease the budget enough to force y_1 or y_2 to become larger than 0. For DRA however, when a busy request arrives at any point, we reset the delay times for u and q back to first values in their respective sequences. This is where DRA increases its cost over BBA, DRA will reset, i.e. increase its wait time after just 1 busy request, where BBA will not increase its wait times, unless there are several requests in which we keep having a negative gain until the budget forces the wait times to starting increasing. In these test inputs, we see that the better choice algorithm is the BBA, especially when the ϵ value for both BBA and DRA are arbitrarily smaller.

11.2 Busy Systems

1^{st} input set	ϵ_{BBA}	BBA	ϵ_{DRA}	DRA	LEA	$\text{Alg}_{1.8}$	OPT
	1/18	101.35	0.1	99.35			
	1/180	96.74	0.01	95.96	84.67	76.58	64.67
	1/1800	95.96	0.001	94.12			
2^{nd} input set	ϵ_{BBA}	BBA	ϵ_{DRA}	DRA	LEA	$\text{Alg}_{1.8}$	OPT
	1/18	108.36	0.1	105.90			
	1/180	102.91	0.01	100.52	87.70	79.03	67.70
	1/1800	99.97	0.001	98.18			
3^{th} input set	ϵ_{BBA}	BBA	ϵ_{DRA}	DRA	LEA	$\text{Alg}_{1.8}$	OPT
	1/18	107.34	0.1	107.38			
	1/180	100.67	0.01	99.07	88.78	79.23	68.78
	1/1800	99.84	0.001	97.38			
4^{rd} input set	ϵ_{BBA}	BBA	ϵ_{DRA}	DRA	LEA	$\text{Alg}_{1.8}$	OPT
	1/18	111.36	0.1	112.85			
	1/180	105.62	0.01	101.75	89.40	80.18	69.40
	1/1800	103.34	0.001	101.74			

Table 11.5: Costs for 3 state algorithms, a = 0.6 and d = 0.4, slack degree 0.25

In the case where the slack degree is 0.25, we see that the LEA and $\text{ALG}_{1.8}$ algorithms have a better performance than DRA and BBA. In the previous examples where BBA had significant savings over DRA, BBA is more costly in several scenarios. As expected with DRA, when the inputs are more busy, DRA is better when ϵ_{DRA} is smaller, also BBA yields a better savings when ϵ_{BBA} is smaller as well. The minimal slack degree that was used in the example trials is 0.25, therefore this input is the busiest input that will be used. Now we will slightly increase the slack degree.

1^{st} input set	ϵ_{BBA}	BBA	ϵ_{DRA}	DRA	LEA	$\text{Alg}_{1.8}$	OPT
	1/18	97.98	0.1	111.57			
	1/180	112.47	0.01	110.68	105.50	95.13	71.50
	1/1800	99.93	0.001	109.82			
2^{nd} input set	ϵ_{BBA}	BBA	ϵ_{DRA}	DRA	LEA	$\text{Alg}_{1.8}$	OPT
	1/18	102.48	0.1	117.37			
	1/180	119.30	0.01	116.70	109.15	97.93	75.15
	1/1800	113.91	0.001	115.81			
3^{rd} input set	ϵ_{BBA}	BBA	ϵ_{DRA}	DRA	LEA	$\text{Alg}_{1.8}$	OPT
	1/18	111.89	0.1	116.58			
	1/180	116.29	0.01	113.11	106.51	96.09	72.51
	1/1800	114.69	0.001	112.59			
4^{th} input set	ϵ_{BBA}	BBA	ϵ_{DRA}	DRA	LEA	$\text{Alg}_{1.8}$	OPT
	1/18	118.51	0.1	117.40			
	1/180	107.19	0.01	112.29	106.61	96.04	72.61
	1/1800	112.30	0.001	111.57			

Table 11.6: Costs for 3 state algorithms, a = 0.6 and d = 0.4, slack degree 0.5

With this slight increase with slack degree, the trends from the earlier section are starting to apply. We see that the costs of the LEA and $\text{ALG}_{1.8}$ have increased and DRA is worse when ϵ_{DRA} is smaller and better when it is larger. The costs of BBA are slightly all similar regardless of its ϵ value, similarly when the input was more slack. LEA and $\text{ALG}_{1.8}$ still have a slightly better output than the tapering down approaches.

1^{st} input set	ϵ_{BBA}	BBA	ϵ_{DRA}	DRA	LEA	$\text{Alg}_{1.8}$	OPT
	1/18	100.15	0.1	118.07			
	1/180	106.68	0.01	117.11	114.68	102.93	74.68
	1/1800	102.39	0.001	116.38			
2^{nd} input set	ϵ_{BBA}	BBA	ϵ_{DRA}	DRA	LEA	$\text{Alg}_{1.8}$	OPT
	1/18	103.44	0.1	113.15			
	1/180	114.53	0.01	112.19	110.26	99.74	71.26
	1/1800	104.86	0.001	111.61			
3^{rd} input set	ϵ_{BBA}	BBA	ϵ_{DRA}	DRA	LEA	$\text{Alg}_{1.8}$	OPT
	1/18	110.21	0.1	115.81			
	1/180	108.56	0.01	115.39	113.66	102.61	73.66
	1/1800	108.60	0.001	116.53			
4^{th} input set	ϵ_{BBA}	BBA	ϵ_{DRA}	DRA	LEA	$\text{Alg}_{1.8}$	OPT
	1/18	107.17	0.1	120.02			
	1/180	100.83	0.01	118.40	115.47	103.92	75.47
	1/1800	110.57	0.001	118.15			

Table 11.7: Costs for 3 state algorithms, a = 0.6 and d = 0.4, slack degree 2/3

We see that the tapering down approaches are not changing as we increase the slack degree, but in comparison they are improving because the cost for LEA and $\text{ALG}_{1.8}$ are increasing. The DRA has better results when the ϵ value is smaller, but the BBA has no pattern, as we change its ϵ value, its cost seems to be rather random.

1^{st} input set	ϵ_{BBA}	BBA	ϵ_{DRA}	DRA	LEA	$\text{Alg}_{1.8}$	OPT
	1/18	113.62	0.1	126.06			
	1/180	112.68	0.01	129.02	125.58	112.38	80.58
	1/1800	129.54	0.001	127.73			
2^{nd} input set	ϵ_{BBA}	BBA	ϵ_{DRA}	DRA	LEA	$\text{Alg}_{1.8}$	OPT
	1/18	105.79	0.1	120.38			
	1/180	106.49	0.01	120.63	122.29	109.98	77.29
	1/1800	103.15	0.001	122.69			
3^{rd} input set	ϵ_{BBA}	BBA	ϵ_{DRA}	DRA	LEA	$\text{Alg}_{1.8}$	OPT
	1/18	102.03	0.1	122.55			
	1/180	102.67	0.01	118.73	119.99	108.17	74.99
	1/1800	120.79	0.001	119.22			
4^{th} input set	ϵ_{BBA}	BBA	ϵ_{DRA}	DRA	LEA	$\text{Alg}_{1.8}$	OPT
	1/18	107.54	0.1	120.14			
	1/180	106.74	0.01	123.00	122.93	110.53	77.93
	1/1800	113.49	0.001	125.67			

Table 11.8: Costs for 3 state algorithms, a = 0.6 and d = 0.4, slack degree 0.8

When the slack degree is closer to 1, DRA has similar behavior to that when the slack degree was above 1, in the previous section, DRA has a better result when ϵ_{DRA} is smaller. BBA in a few inputs sets it has similar cost regardless of its ϵ value and other runs it has better results when its ϵ is larger. Similar to DRA, since the system is more slack than before, it has a smaller cost when its ϵ value is larger. For all the cases of when the input had a slack degree below one, $\text{ALG}_{1.8}$ always yielded better output than any of the other online algorithms. As expected, the taper down approaches are better when we have slack input not busy input, we saw this with the 2 state problem and here with the 3 state problem. The DRA was not optimal in a busy system but had slightly better cost for busier systems than BBA, when the slack degree was approaching 1 and when the slack degree was greater than 1, BBA was more optimal than the DRA.

Chapter 12

Conclusion

12.1 Summary

In this dissertation, we analyzed several models of the power down problem. We first presented the fundamentals of online algorithms. An online algorithm is such an algorithm that makes its decisions without any knowledge of the future. Online algorithms played a key role for power down problems because we needed to model a system for a real-world application where future requests cannot be known. We used online competitive analysis because we want to model our algorithm after the worst case, such that we know how our algorithm would perform in the worst case which is why we avoided queueing theory, if we our algorithm could perform well in the worst case, it would imply that it would perform well in the best case, or at least better than it did in the worst case. We then began our work from the two state problem which is based on the ski rental algorithm which was proven to be a 2-competitive algorithm.

From the two state problem, we introduced the three state problem. In this problem, we had some intermediate state that allowed the machine to switch from the ON state to this intermediate state rather than switch straight to the OFF state. Adding in this third state overall decreased the competitive ratio, as with the two state problem, we needed to choose a switch time that would yield a minimal competitive ratio. Once we choose that switch time, we yielded a better competitive ratio than 2. Also when if we set the unit cost to be 0.6 times the ON state unit cost and 0.4 times the OFF state power up cost, we obtained the minimum competitive ratio for the three state problem of 1.8.

We then build on the three state principle to develop an algorithm for the five state system. As with the three state system, the five state system would have three intermediate states which can be used alternatively than switching from ON to OFF. We developed an approximation algorithm that

computes the switch times to reach a target competitive ratio and using a binary search technique of possible competitive ratios in a range until we reach the minimal competitive ratio within some approximation. This method was based on the method used for the n state power machine but since it was concentrated specifically on the five state machine, we were able to introduce this simple version of obtaining a schedule that yielded the minimum competitive ratio for the five state system.

After working on systems with small number of states, and since there was prior work done on n state machines, we made research on the infinite state problem, and we focused our attention from the discrete model to this continuous model. Similar to the discrete model, we assigned a unit cost and a power up cost to each state in this continuous range by creating two continuous functions, one for the unit cost and the other for the power up cost. Once again, similar to any power down system, we needed to devise a schedule to attempt to obtain a minimal competitive ratio. We first tried the lower envelope approach which gave us a 2-competitive result. We obtained the optimal offline algorithm transition schedule, and then we developed a few different mechanisms to transition to the lower power states that were used in the online algorithm. We could classify these strategies as transitioning to lower power states faster than the offline and slower than the offline. After experimenting with these strategies were able to conclude that the better strategy were the ones that transitioned to lower power states faster than then the offline algorithm strategy.

We then introduced two tapering based strategies. The first one was the decrease and reset algorithm (DRA). We classified a request as either busy or slack, and when the system had a set of consecutive slack requests, the machine began to tapering down, since the requests arriving in this fashion suggested that the machine was not busy and thus could turn itself off at an earlier time to save energy. Using this method allowed us to change the wait times to save energy while not increasing the competitive ratio by a large factor. We then presented another technique to taper down the wait times which uses a budget, the amount of savings that the algorithm had dictated how the wait time adjusted while also not increasing the competitive ratio by a large factor either. The difference between DRA and the budget based technique is that DRA would only taper down on a slack request and as soon as a busy request arrived, then it would adjust back to the maximum wait time. where the budget based technique would taper down only when energy saved (which occurred with a slack request but not limited to a slack request), and would adjust its wait time to a larger amount only when budget decreased (if energy decreased).

Then we compared the two state DRA with the budget based algorithm along with the OWCR with a set of test inputs. The budget based algorithm performed better than the DRA as the slack degree increased for the input. Both algorithms had a control parameter ϵ that causes the competitive ratio to increase and controls how the algorithms adjust their wait times. When we

set ϵ to a higher value, both techniques performed better than OWCR, however the budget based technique out performed DRA using any ϵ value for any slack input. When we experimented on busy input, we saw that neither the DRA not budget based technique performed better than the OWCR. These tapering based techniques adjust their wait time based on saving energy or on consecutive slack requests, and thus these tapering strategies produce favorable results when we have many instances of long wait times between requests.

We then apply these tapering down strategies onto the three state power down problem. In this case we tapered down the wait time in the ON state as well as the INT state, however, the DRA only adjusted its ON state wait duration and not the INT state duration after consecutive slack requests. Once the ON state duration became 0, only then did the INT state duration begin to taper down and often it would take one or two slack requests to taper the INT state wait time down to 0. With the budget based technique we were able to simultaneously taper the ON state and INT state durations. We once again ran simulations with these two tapering algorithms along with the lower envelope algorithm and the three state technique that have a 1.8-competitive result. Once again, we saw better results with the tapering strategies for a higher slack degree input and especially when ϵ was set to a higher value. Once again, the budget based algorithm out performed DRA in the three state setting. However, if the input was busy, the DRA and budget based technique performed worse than the non tapering algorithms.

This dissertation was done to investigate the power down problem in several isolated environments from few states to infinite states to analyze the behavior in great detail in each environment. We were able to reach an upper bound on the three state state system to be 1.8-competitive without using an approximation to bound this result. We came to the conclusion that having a machine with more states yield a better competitive ratio, which was shown on the five state problem and the infinite state problem. We also reached results on the taper down based algorithms which showed significantly better results for experimental input with only a slight increase in the competitive ratio, and this technique can be applied to any system with any number of states.

12.2 Future Work

Our future work will consist of refining and combing our ideas from this thesis. The work done on the infinite state problem is quite sparse, since not much work has been done on that subject. We will continue to develop new strategies to attempt to come up with upper or lower bounds, which also includes coming up with upper or lower bounds for the three and five state power down machine. Since the tapering down strategy is a generic strategy that can be applied to any model, we can consider applying the budget based technique and the DRA onto the five state machine as

well as the infinite state problem. We can also extend the power down problem to a distributed system or to a multiple machine or multicore environment as well. In a distributed system, there are often times when a node is in some idle state or spinlocking and so putting the node into a lower power state or some sleep state can be beneficial. We can apply these problems to a multi threaded environments since they are widely used in web development and smart phone development and thus investigating these problems in multicore and/or in a multi threaded environment can lead to favorable results, since multi-threaded and multicore environments can spend time busy waiting for an event to occur and thus the power down problem is directly applicable in this scenario for example when a thread is waiting long and putting the thread to sleep after limited activity can also lead to favorable results. The power down problem is an applicable problem in many areas, even beyond the area of information technology, and we seek to gain a great deal of understanding of this problem and how it can be used in many areas of study.

Bibliography

[1] https://msdn.microsoft.com/en-us/library/windows/desktop/aa373229(v=vs.85).aspx, 2016.

[2] https://support.apple.com/en-us/HT201714, 2016.

[3] Yuvraj Agarwal, Steve Hodges, Ranveer Chandra, James Scott, Paramvir Bahl, and Rajesh Gupta. Somniloquy: Augmenting network interfaces to reduce pc energy usage. In *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation*, NSDI'09, pages 365–380, Berkeley, CA, USA, 2009. USENIX Association.

[4] Yuvraj Agarwal, Stefan Savage, and Rajesh Gupta. Sleepserver: A software-only approach for reducing the energy consumption of pcs within enterprise environments. In *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, USENIXATC'10, pages 22–22, Berkeley, CA, USA, 2010. USENIX Association.

[5] Susanne Albers. *Online Algorithms*, pages 143–164. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[6] Susanne Albers. Energy-efficient algorithms. *Communications Of The ACM*, 53:86–96, 2010.

[7] C. Anderson and A. R. Karlin. Two adaptive hybrid cache coherency protocols. In *High-Performance Computer Architecture, 1996. Proceedings., Second International Symposium on*, pages 303–313, Feb 1996.

[8] J. Andro-Vasko, W. Bein, D. Nyknahad, and H. Ito. Evaluation of online power-down algorithms. In *2015 12th International Conference on Information Technology - New Generations*, pages 473–478, April 2015.

[9] John Augustine, Sandy Irani, and Chaitanya Swamy. Optimal power-down strategies. In *IEEE Symposium on Foundations of Computer Science*, pages 530–539. Cambridge University Press, 2004.

[10] Y. Azar, Y. Bartal, E. Feuerstein, A. Fiat, S. Leonardi, and A. Rosén. On capital investment. *Algorithmica*, 25(1):22–36, 1999.

[11] N. Bansal, H. L. Chan, T. W. Lam, and K. L. Lee. Scheduling for speed bounded processors. In *Proc. 35th International Colloquium on Automata, Languages and Programming*, pages 409–420, 2008.

[12] N. Bansal, H. L. Chan, K. Pruhs, and D. Katz. Improved bounds for speed scaling in devices obeying the cube-root rule. In *roc. 36th International Colloqium on Automata, Languages and Programming*, pages 144–155, 2009.

[13] L.A. Barroso. The price of performance. *ACM Queue 3*, 2005.

[14] Wolfgang Bein, Naoki Hatta, Nelson Hernandez-Cons, Hiro Ito, Shoji Kasahara, and Jun Kawahara. An online algorithm optimally self-tuning to congestion for power management problems. In *Proceedings of the 9th International Conference on Approximation and Online Algorithms*, pages 35–48. Springer-Verlag, 2012.

[15] L. A. Belady. A study of replacement algorithms for a virtual-storage computer. *IBM Syst. J.*, 5(2):78–101, June 1966.

[16] S. Ben-David, A. Borodin, R. Karp, G. Tardos, and A. Wigderson. On the power of randomization in online algorithms. In *Proceedings of the Twenty-second Annual ACM Symposium on Theory of Computing*, STOC '90, pages 379–386, New York, NY, USA, 1990. ACM.

[17] L. Benini, A. Bogliolo, and G. De Micheli. A survey of design techinques for system-level dynamic power management. *IEEE Trans. VLSI Syst.*, 8:299–316, 2000.

[18] Allan Borodin, Prabhakar Raghavan, Sandy Irani, and Baruch Schieber. Competitive paging with locality of reference. In *Proceedings of the Twenty-third Annual ACM Symposium on Theory of Computing*, STOC '91, pages 249–259, New York, NY, USA, 1991. ACM.

[19] Joan Boyar, Susan Krarup, and Morten N. Nielsen. Seat reservation allowing seat changes. *J. Algorithms*, 52(2):169–192, August 2004.

[20] Marshall Brain. http://www.science.smith.edu/ jcardell/Courses/EGR220/ElecPwr_HSW.html.

[21] Eui-Young Chung, L. Benini, A. Bogliolo, Yung-Hsiang Lu, and G. De Micheli. Dynamic power management for nonstationary service requests. *IEEE Transactions on Computers*, 51(11):1345–1361, Nov 2002.

[22] Peter Damaschke. Nearly optimal strategies for special cases of on-line capital investment. *Theoretical Computer Science*, 302(1):35 – 44, 2003.

[23] S. Ben David and A. Borodin. A new measure for the study of online algorithms, 2001.

[24] Pierre Delforge. America's data centers consuming and wasting growing amounts of energy. *National Resources Defense Council*, 2015.

[25] S. J. Eggers and R. H. Katz. Evaluating the performance of four snooping cache coherency protocols. *SIGARCH Comput. Archit. News*, 17(3):2–15, April 1989.

[26] X. Fang, S. Misra, G. Xue, and D. Yang. Smart grid - the new and improved power grid: A survey. *IEEE Communications Surveys Tutorials*, 14(4):944–980, Fourth 2012.

[27] Amos Fiat, Richard M. Karp, Michael Luby, Lyle A. McGeoch, Daniel Dominic Sleator, and Neal E. Young. Competitive paging algorithms. *CoRR*, cs.DS/0205038, 2002.

[28] X. Han, T. W. Lam, L. K. Lee, I. K. K. To, and P. W. H. Wong. Deadline scheduling and power management for speed bounded processors. *Theoretical Computer Science*, 411:3587–3600, 2010.

[29] S. Irani, R. Gupta, and S.K. Shukla. Algorithm for power savings. *ACM Transactions on algorithms*, 3, 2007.

[30] Sandy Irani, Rajesh Gupta, and Sandeep Shukla. Online strategies for dynamic power management in systems with multiple power-saving states. *ACM Transactions on Embedded Computing Systems*, 2(3), 2003 2003.

[31] Sandy Irani and Anna R. Karlin. On online computation. In *Approximation Algorithms for NP-Hard Problems, chapter 13*, pages 521–564. PWS Publishing Company, 1997.

[32] Sandy Irani and Kirk R. Pruhs. Algorithmic problems in power management. *SIGACT News*, 36(2):63–76, June 2005.

[33] Sandy Irani, Sandeep Shukla, and Rajesh Gupta. Online strategies for dynamic power management in systems with multiple power-saving states. *ACM Trans. Embed. Comput. Syst.*, 2(3):325–346, August 2003.

[34] Sandy Irani and Gaurav Singh. An overview of the competitive and adversarial approaches to designing dynamic power management strategies. *IEEE Transactions Very Large Scale Integration*, 13(12), December 2005.

[35] Aman Kansal Jitendra Padhye Jitu Padhye Joshua Reich, Michel Goraczko. Sleepless in seattle no longer. Technical report, March 2010.

[36] Anna R. Karlin, Claire Kenyon, and Dana Randall. Dynamic tcp acknowledgement and other stories about e/(e-1). In *Proceedings of the Thirty-third Annual ACM Symposium on Theory of Computing*, STOC '01, pages 502–509, New York, NY, USA, 2001. ACM.

[37] Anna R. Karlin, Kai Li, Mark S. Manasse, and Susan Owicki. Empirical studies of competitve spinning for a shared-memory multiprocessor. *SIGOPS Oper. Syst. Rev.*, 25(5):41–55, September 1991.

[38] Anna R. Karlin, Mark S. Manasse, Larry Rudolph, and Daniel D. Sleator. Competitive snoopy caching. In *Proceedings of the 27th Annual Symposium on Foundations of Computer Science*, SFCS '86, pages 244–254, Washington, DC, USA, 1986. IEEE Computer Society.

[39] A.R. Karlin, L.A. McGeogh, and S.S. Owicki. Competitive randomized algorithms for non uniform problems. *Algorithmica 11*, pages 542–571, 1994.

[40] Claire Kenyon. Best-fit bin-packing with random order. In *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '96, pages 359–364, Philadelphia, PA, USA, 1996. Society for Industrial and Applied Mathematics.

[41] S. Keshav, C. Lund, S. Phillips, N. Reingold, and H. Saran. An empirical evaluation of virtual circuit holding time policies in ip-over-atm networks. *IEEE Journal on Selected Areas in Communications*, 13(8):1371–1382, Oct 1995.

[42] Gunjan Kumar and Saswata Shannigrahi. New online algorithm for dynamic spped scaling with sleep state. Technical report.

[43] Zvi Lotker, Boaz Patt-Shamir, and Dror Rawitz. Rent, lease or buy: Randomized algorithms for multislope ski rental. *CoRR*, abs/0802.2832, 2008.

[44] Mark Manasse, Lyle McGeoch, and Daniel Sleator. Competitive algorithms for on-line problems. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, pages 322–333, New York, NY, USA, 1988. ACM.

[45] Sergiu Nedevschi, Jaideep Chandrashekar, Junda Liu, Bruce Nordman, Sylvia Ratnasamy, and Nina Taft. Skilled in the art of being idle: Reducing energy waste in networked systems. In *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation*, NSDI'09, pages 381–394, Berkeley, CA, USA, 2009. USENIX Association.

[46] Konstantinos Panagiotou and Alexander Souza. On adequate performance measures for paging. In *Proceedings of the Thirty-eighth Annual ACM Symposium on Theory of Computing*, STOC '06, pages 487–496, New York, NY, USA, 2006. ACM.

[47] S. Phillips and J. Westbrook. *Algorithms and Theory Computation Handbook*. CRC Press, 1999.

[48] D. Ramanathan, S. Irani, and R. Gupta. Latency effects of system level power management algorithms. In *Computer Aided Design, 2000. ICCAD-2000. IEEE/ACM International Conference on*, pages 350–356, Nov 2000.

[49] Nick Reingold, Jeffery Westbrook, and Daniel D. Sleator. Randomized competitive algorithms for the list update problem. *Algorithmica*, 11(1):15–32, 1994.

[50] Daniel D. Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28(2):202–208, February 1985.

[51] F.F. Yao, A.J. Demers, and S. Shenker. A scheduling model for reduced cpu energy. In *Proc. 36th IEEE Symposium on Foundations of Computer Science*, pages 374–382, 1995.

Curriculum Vitae

Graduate College
University of Nevada, Las Vegas

James Andro-Vasko
Email: androvas@unlv.nevada.edu

Degrees:

Master of Science in Computer Science 2011
University of Nevada Las Vegas
Bachelor of Science in Computer Science 2009
University of Nevada Las Vegas

Dissertation Title: Competitive Power Down Methods In Green Computing

Thesis Examination Committee:

Chairperson, Dr. Wolfgang Bein, Ph.D.
Committee Member, Dr. Lawrence Larmore, Ph.D.
Committee Member, Dr. Ajoy Datta, Ph.D.
Committee Member, Dr. Andreas Setfik, Ph.D.
Graduate Faculty Representative, Dr. Robert Boehm, Ph.D.