2011

Data-aware workflow scheduling in heterogeneous distributed systems

Dengpan Yin
Louisiana State University and Agricultural and Mechanical College, yindengpan@gmail.com

DATA-AWARE WORKFLOW SCHEDULING IN HETEROGENEOUS DISTRIBUTED SYSTEMS

A Dissertation

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

in

The Department of Computer Science

by
Dengpan Yin
B.E., Nanjing University of Aeronautics and Astronautics, China, 2006
M.S., Southern University, Baton Rouge, Louisiana, 2008
December, 2011

Acknowledgments

First and foremost I would like to gratefully thank my supervisor and dissertation chair Dr. Tevfik Kosar. I appreciate all his contributions of ideas, time and funding to make my research productive and stimulating. His patience, encouragement, supervision and support enabled me to develop a deep understanding of the research problem. At the same time, I would like to gratefully thank Dr. Gabrielle Allen. I appreciate her very much for providing me a chance to participate in the NGCHC project as a research assistant during my last semester.

Meanwhile, I am heartily thankful to my advisor and dissertation chair Dr. Konstantin Busch. His valuable advise, suggestions and guidance greatly helped me during my research. Also, I would like to acknowledge my committee members: Dr. Jianhua Chen, Dr. Bijaya B. Karki, Dr. Gerald Baumgartner and Dr. Guoxiang Gu for their time, interest and helpful comments.

Furthermore, it is my pleasure to thank CCT (Center for Computation and Technology) for providing excellent research environment. I sincerely thank my colleagues in CCT for insight discussions during my research.

Lastly, I would like to thank my family for their love and encouragement. With their encouragement and patience, I would be able to face the tough times in my Ph. D. pursuit.

Table of Contents

List of Tables

List of Figures

Abstract

Data transferring in scientific workflows gradually attracts more attention due to large amounts of data generated by complex scientific workflows will significantly increase the turnaround time of the whole workflow. It is almost impossible to make an optimal or approximate optimal scheduling for the end-to-end workflow without considering the intermediate data movement. In order to reduce the complexity of the workflow-scheduling problem, most researches done so far are constrained by many unrealistic assumptions, which result in non-optimal scheduling in practice. A constraint imposed by most researchers in their algorithms is that a computation site can only start the execution of other tasks after it has completed the execution of the current task and delivered the data generated by this task. We relax this constraint and allow overlap of execution and data movement in order to improve the parallelism of the tasks in the workflow. Furthermore, we generalize the conventional workflow to allow data to be staged in(out) from(to) remote data centers, design and implement an efficient data-aware scheduling strategy. The experimental results show that the turnaround time is reduced significantly in heterogeneous distributed systems by applying our scheduling strategy.

To reduce the end-to-end workflow turnaround time, it is crucial to deliver the input, output and intermediate data as fast as possible. However, it is quite often that the throughput is much lower than expected while using single TCP stream to transfer data when the bandwidth of the network is not fully utilized. Multiple TCP streams will benefit the throughput. However, the throughput does not increase monotonically when increasing the number of parallel streams. Based on this observation, we propose to improve the existing throughput prediction models, design and implement a TCP throughput estimation and optimization service in the distributed systems to figure out the optimal configurations of TCP parallel streams. Experimental results show that the proposed estimation and optimization service can predict the throughput dynamically with high accuracy and the throughput can be increased significantly. Throughput optimization along with data-aware workflow scheduling allows us to minimize the end-to-end workflow turnaround time successfully.

Chapter 1
Introduction

1.1 Preliminaries of Data-aware Scheduling

With the rapid deployment of Grid infrastructures and the collaborations between these different Grid organizations, such as LONI and TeraGrid [38], it becomes feasible and promising to run scientific workflow applications on these large scale distributed infrastructures. A workflow is a group of tasks with interdependencies. After the execution of one task in the workflow, it will produce some data that will be used as input data of one or more tasks in the workflow. Also one task might need data produced by more than one tasks. We call the task that produces data a predecessor of the tasks that consume the data, and call the task that consumes the data a successor of the tasks that produce the data. Each task might have more than one successor or predecessor.

In the cloud computing paradigm [20] [17], a user can apply for a certain amount of computing resources dedicated for its application. The user is responsible to pay the cloud provider according to the amount of computing resources and the service time. There are several successful cloud computing platforms such as Amazon EC2 [1] and Microsoft Azure [5]. When a workflow application is submitted to the cloud, the user expects a shortest execution time for the whole workflow. The workflow scheduler will dispatch each individual task to a certain site for execution and the intermediate data will move around the data cloud through the network. Hence it is important to fully utilize the available bandwidth to shorten the data placement time. Furthermore, different scheduling schemes will result in different turnaround time of the workflow. To be cost effective, an optimal or approximate optimal scheduling is essential.

A workflow comprises many small tasks, of which the independent tasks can be dispatched to different computation sites in the distributed systems and executed in parallel. Once the workflow is submitted to the Grid or Cloud system, it will be executed automatically without the user's intervention. With the transparency, the user avoids manually operating on each individual task. Data intensive workflows have been applied in many fields such as astronomy [14], bioinformatics [37] and high-energy physics [33]. In these applications, terabytes of data will be processed by the workflow and some important intermediate data need to be stored for future use. Workflow scheduling problem has been studied for decades; however, to our best knowledge, none of the existing algorithms can give an optimal solution for the data intensive workflows. Along with the increasing data size in the workflow applications, it is imperative for us to develop a new algorithm to address this scheduling problem with an optimal approach.

To address the data-aware workflow-scheduling problem, we need a sophisticated algorithm to take care of the dependencies of these tasks. Each task must be scheduled before its successors and after its predecessors. At a specific moment, there might be more than one ready task, if the number of computational sites are less than the number of ready tasks, the order of scheduling them will affect the turnaround time of the workflow. Meanwhile, for each ready task, there will be more than one computational site that can be mapped to. Due to the heterogeneous nature of the network bandwidth and the computation power, different mapping schemes will affect the end-to-end turnaround time.

1.2 Data Throughput of Workflow

In a widely distributed data-aware computing paradigm, data delivery between participating computation sites may become a major performance bottleneck [22]. An optimized TCP throughput will definitely benefit the data-aware workflow scheduling algorithm. Today, many regional and national optical networking initiatives such as LONI [3], ESnet [2] and Teragrid [4] provide high speed network connectivity to their users. However, it is quite often that users cannot fully utilize the available bandwidth due to misconfigurations.

The end-to-end performance of a data transfer over the network depends heavily on the underlying network protocol used. TCP is the most widely adopted transport protocol, however, its AIMD property, which aims to maintain fairness among streams sharing the network, prevents TCP to fully utilize the available network bandwidth. This becomes a major problem, especially for wide area network (WAN) with high latency. There have been different implementation techniques both in the kernel and application levels to overcome the poor network utilization of TCP. In the transport layer, different variations of TCP have been implemented [16, 24, 26] to utilize high-speed networks, but there is not a single adopted protocol to replace the regular TCP. At the application level, opening parallel streams is one way of doing that and is widely used in many application areas.

Parallel streams are able to achieve high throughput by behaving like a single large stream that is the combination of n streams, and can get an unfair share of the available bandwidth [8, 15, 19, 25, 29, 31, 41]. However, using too many streams can bring the network to a congestion point very easily, especially for low-bandwidth networks. It is important to find the optimal parallelism level where the network is saturated and the throughput remains stable. Unfortunately, it is difficult to predict this optimal point.

In this research, a service is proposed to provide the data-aware workflow scheduler with the optimal parallel stream number and a provision of the estimated time and throughput information for a specific data transfer. The optimal stream number is calculated using a novel mathematical model that we have developed in [49]. With this estimation and optimization service, data can be delivered much faster.

1.3 Contributions

In this research, a data-aware scheduling algorithm is proposed for the data intensive workflows. A throughput estimation and optimization service is implemented as an integral part of the data intensive workflow scheduling. More specifically, the major contributions include the following:

- Design a novel queue based data-aware model for data intensive workflows and optimize the turnaround time by allowing overlap of task execution and data placement. The turnaround time is reduced and the time efficiency of the algorithm is improved significantly.
- Design and implement a data-aware co-scheduling algorithm by considering external data, intermediate data and computing for both homogeneous and heterogeneous distributed systems. The turnaround time is reduced in both scenarios.
- Improve a TCP throughput prediction model and prove the correctness experimentally and theoretically. The precision is the best compared with the existing models.

- Design and implement a throughput estimation and optimization service (EOS) in the distributed environments for the data-aware workflow scheduling. The EOS service can make the workflow fully utilize the available bandwidth and the throughput is improved significantly.

Chapter 2
Background and Related Works

The workflow scheduling problem has been well studied for many years and today it is still a very active research area. In order to minimize the turnaround time, many approximation algorithms have been proposed, such as genetic algorithms [45, 47, 51], simulated annealing algorithms [50] and ant colony algorithms [36]. There are some common features of these algorithms. Even through they do not guarantee an optimal solution, they guarantee to generate an acceptable solution in a timely manner. The quality of the solution is controlled by a series of parameters.

There are also several approaches that aim to find the optimal solution of the workflow scheduling problem. In [10] Chou and Chung proposed to find the optimal scheduling for workflows on multiprocessors. However, the communication cost is ignored due to low latency between processors. In [9], Chang and Jiang proposed a state space search algorithm to address the problem. They used the critical path length as an underestimate of the actual cost function to guide the expansion of the state during the solution exploring process. In their research, the communication cost between tasks is also ignored. Kwok and Ahmad [28]proposed a parallel state space search approach based on A-Star algorithm. Also they applied state-pruning techniques to reduce the search space. In their work, they assumed that the bandwidth between different processors are homogeneous.

Wang and Tsai [46] proposed a state space search approach based on A-star algorithm to solve the workflow scheduling problem in a distributed system. In their research, they assumed that the computation power of the computational sites and the network bandwidth between them are heterogeneous. They claimed that their solution to be optimal. Lin [30] studied the same problem and pointed out that Wang's solution is not optimal in some cases. Wang and Tsai assumed that one of the immediately preceding communications of a task must be performed just before the execution of the task. Lin relaxed this constraint and rearranged the task execution and task communication orders in order to get the optimal scheduling.

Both Lin and Wang et al have assumed that the computation sites should be in either execution state or communication state. It cannot overlap execution and data transfer. With this constraint, the problem can be simplified, however, the scheduling turns out to be non-optimal in some cases. It makes sense that a computation site can receive input data for a task when it is executing another task. The data will be ready at the execution finish time of the previous task. Based on this investigation, we further remove that constraint and decrease the turnaround time by overlapping the execution and data placement.

In all of these previous works mentioned above, the authors paid little attention to the data placement. None of them mentioned about how to deal with the scheduling problem when there are extra data that need to be staged in from remote site or to be staged out to a remote site for a task.

To our best knowledge, most of works done so far on this research fall on non-optimal solutions [34, 35].We extend the state space search algorithm to find an optimal scheduling for the data intensive workflows. Our experiments show that the turnaround time of the workflow can be reduced significantly when the network bandwidth is heterogeneous.

The studies that try to find the optimal number of streams are so few and they are mostly based on approximate theoretical models [7,11,18,27,32]. They all have specific constraints and assumptions. Also the correctness of the proposed models are mostly proved with simulation results only. Hacker et al. claim that the total number of streams behaves like one giant stream that transfers in capacity of total of each streams' achievable throughput [18]. However, this model only works for uncongested networks. Thus, it cannot provide a feasable solution for congested networks. Another study [11] declares the same theory but develops a protocol which at the same time provides fairness. Dinda et al. [32] model the bandwidth of multiple streams as a partial second order equation and require two different throughput measurement of different stream numbers to predict the others. However, this model cannot predict the optimal number of parallel streams necessary to achieve best transfer throughput. In another model [7], the total throughput always shows the same characteristics depending on the capacity of the connection as the number of streams increases and 3 streams are sufficient to get a 90% utilization. A new protocol study [27] that adjusts sending rate according to calculated backlog presents a model to predict the current number of flows which could be useful to predict the future number of flows.

All of the models presented have either poor accuracy or they need a lot of information to be collected. Unfortunately, users do not want to present this information or have no idea what to supply to a data transfer tool. They need a means to make a projection of their data transfer throughput and must gather the information to optimize their transfer without caring about the characteristics of an environment and the transfer at hand. For individual data transfers, instead of relying on historical information, the transfers should be optimized based on instant feedback. In our case, this optimization is achieving optimal number of parallel streams to get the highest throughput. However, an optimization technique not relying on historical data in this case must not cause overhead of gathering instant data that is larger than the speed up gained with multiple streams for a particular data size. Gathering instant information for prediction models could be done by using network performance measurement tools [23, 39, 40, 42, 48] or doing a miniature version of the transfer.

In the proposed service, Iperf [48] or GridFTP [6] is used to gather the sampling information to be fed into the mathematical models. Both of the tools are widely adopted by the Grid community and convenient for our service since they both support parallel streams. With GridFTP, it is also very convenient to perform third-party transfers. By using the mathematical models and the instant sampling information, the proposed service will give the optimal parallel stream number with a negligible prediction cost.

Chapter 3
Data Intensive Workflow

Task scheduling problem has become a more and more important issue with the advance of distributed computing research. As new computation paradigms emerge such as many-task computing and cloud computing, the scheduling problem has become a major bottleneck in improving the performance of the proposed system.

The problem of interest can be described as the following. We are given a group of tasks with dependencies, the data sites for each individual task, the computation power of each computation site in the heterogeneous system and the capacities of each link connecting these sites, the objective is to minimize the turnaround time of these group of tasks considering how to assign these tasks to different computation sites, in which order these tasks should be executed, and how the data flow between these tasks should be scheduled.

3.1 Distributed Systems Notation

Let N_c be the number of computation sites. N_d is the number of data sites. N_t is the number of task modules. P is a set of computation sites p_i. $P = \{p_i\}$ $i = 1, 2, \cdots, N_c$. D is a set of data sites of d_i. $D = \{d_i\}$ $i = 1, 2, \cdots, N_d$. C is a set of computation links between computation sites $c_{p_i p_j}$, data sites $c_{d_k d_l}$, and computation-data sites $c_{p_i d_k}$. $C = \{c_{p_i p_j}, c_{d_k d_l}, c_{p_i d_k}\}$ $i, j = 1, 2, \cdots, N_c; k, l = 1, 2, \cdots, N_d$. S is a distributed system which consists of P, D, C. $S = (P, D, C)$. A typical distributed system of interest can be illustrated in Figure 3.1.

3.2 Conventional Workflow Notation

Let W represent the workflow which consists of a group of tasks. $W = \{t_i\}$. Some of these tasks have dependence relationship and the workflow is demoted by a directed acyclic graph (DAG). Task t_i is immediately dependent on task t_j if the output of t_j is the input of t_i. The immediate predecessor of t_i is a set of tasks on which t_i is immediately dependent on. Use $Pre(t_i)$ to denote the immediate

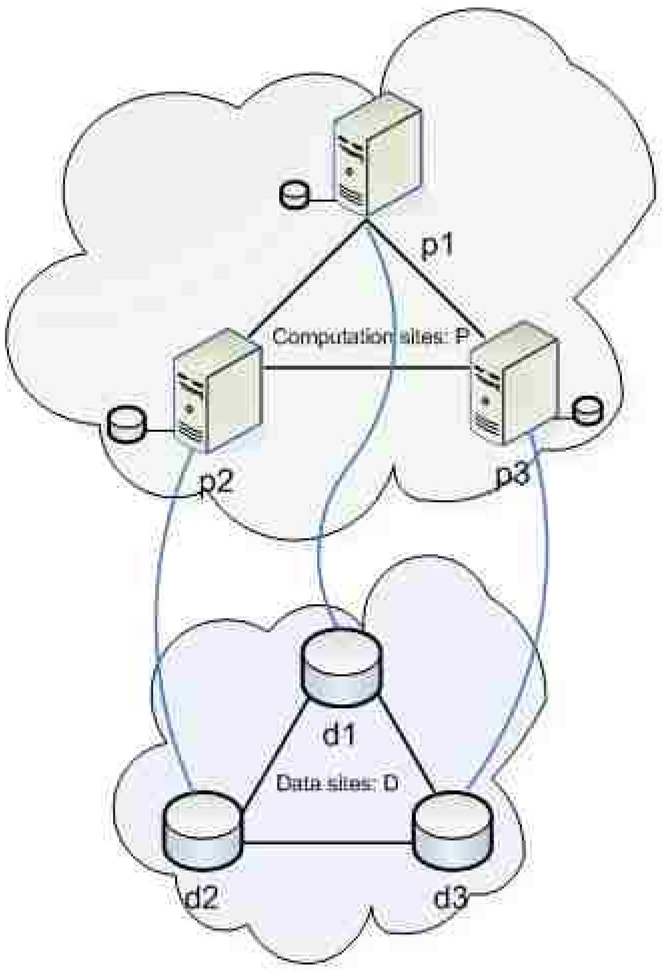


FIGURE 3.1: Distributed systems

predecessor. The immediate successor of t_i is a set of tasks which are immediately dependent on t_i. Use $Suc(t_i)$ to denote the immediate successor of t_i. Clearly if $t_i \in Pre(t_j)$, then $t_j \in Suc(t_i)$.

3.3 Extended Workflow Notation

The conventional workflow only consists of a group of tasks and the dependencies between them. Besides the data needed from the output of dependent tasks, one task might need some extra data stored in remote storage sites. In order to execute the task, the extra data might be staged in to the local cache before execution of the current task. Also the output of some tasks might be important so that it should be staged out to remote storage sites after execution. We extend the conventional workflow to consist of stage in and stage out phases. The immediate predecessor of a task t_i is the union of the predecessor of t_i in the conventional workflows and a set of data sites from which extra data should be staged in before execution. Use $ExPre(t_i)$ to denote the immediate predecessor of t_i. $Expre(t_i) =$

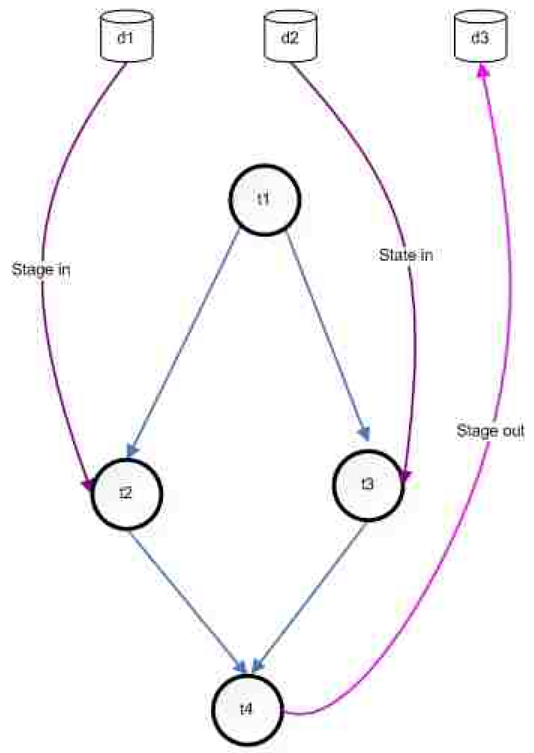


FIGURE 3.2: Extended workflow

$Pre(t_i) \cup \{d_i : d_i\ stores\ data\ should\ be\ staged\ in\ before\ execution\ of\ t_i\}$. Use $ExSuc(t_i)$ to denote the immediate successor of t_i. $ExSuc(t_i) = Suc(t_i) \cup \{d_i : d_i\ stores\ data\ should\ be\ staged\ out\ after\ execution\ of\ t_i\}$. The extended workflow can be depicted by Figure 3.2.

3.4 Constraints and Assumptions

1) A task can only be executed after it obtained all the data required from remote data sites and all the intermediate data from the computational sites where tasks belonging to its predecessors are executed.

2) Each task consists of two phases, one of which is computation and the other is intermediate data transfer. Once a task starts execution, it cannot be stopped until the computation work is done. Once an intermediate data transfer is started, it cannot stop until this transfer is finished.

3) The intermediate data transfer is not necessary to be started immediately after the computation work is done.

4) A computation site can execute another ready task t_j once the current task t_i finishes its computational phase. The computational phase of t_j can be overlapped with the intermediate data transfer phase of task t_i.

3.5 Objective Function of the Scheduling

Each task t_i needs a finite amount of time t_{ij} to finish if it can be scheduled to computation site p. Define

$$EXT(t_i, p_j) = \begin{cases} t_{ij} & t_i \textit{ is applicable on } p_j \\ \infty & \textit{otherwise} \end{cases}$$

Define the task computation matrix on the computational sites as:

$$\begin{aligned} M_c &= \begin{pmatrix} EXT_{11} & EXT_{12} & \cdots & EXT_{1N_c} \\ EXT_{21} & EXT_{22} & \cdots & EXT_{2N_c} \\ \cdots & \cdots & \cdots & \cdots \\ EXT_{N_t1} & EXT_{N_t2} & \cdots & EXT_{N_tN_c} \end{pmatrix} \\ &= [EXT_{ij}]_{1\le i\le N_t, 1\le j\le N_c} \end{aligned}$$

Use TH_{ij} to denote the throughput between sites i and j. Define the throughput matrix between the computational sites as:

$$\begin{aligned} M_t &= \begin{pmatrix} TH_{11} & TH_{12} & \cdots & TH_{1N_c} & \cdots & TH_{1(N_c+N_d)} \\ TH_{21} & TH_{22} & \cdots & TH_{2N_c} & \cdots & TH_{2(N_c+N_d)} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ TH_{(N_c+N_d)1} & TH_{(N_c+N_d)2} & \cdots & TH_{(N_c+N_d)N_c} & \cdots & TH_{(N_c+N_d)(N_c+N_d)} \end{pmatrix} \\ &= [TH_{ij}]_{1\le i\le N_c+N_d, 1\le j\le N_c+N_d} \end{aligned}$$

Use IMD_{ij} to denote the intermediate data between tasks t_i and t_j. Define intermediate data matrix as:

$$
\begin{aligned}
M_{imd} &= \begin{pmatrix} IMD_{11} & IMD_{12} & \cdots & IMD_{1N_t} \\ IMD_{21} & IMD_{22} & \cdots & IMD_{2N_t} \\ \cdots & \cdots & \cdots & \cdots \\ IMD_{N_t1} & IMD_{N_t2} & \cdots & IMD_{N_tN_t} \end{pmatrix} \\
&= [IMD_{ij}]_{1\le i\le N_t, 1\le j\le N_t}
\end{aligned}
$$

Use SID_{ij} to denote the stage in data from data site d_j to task t_i. Define stage in data matrix as:

$$
\begin{aligned}
M_{sid} &= \begin{pmatrix} SID_{11} & SID_{12} & \cdots & SID_{1N_d} \\ SID_{21} & SID_{22} & \cdots & SID_{2N_d} \\ \cdots & \cdots & \cdots & \cdots \\ SID_{N_t1} & SID_{N_t2} & \cdots & SID_{N_tN_d} \end{pmatrix} \\
&= [SID_{ij}]_{1\le i\le N_t, 1\le j\le N_d}
\end{aligned}
$$

Use SOD_{ij} to denote the stage out data from task t_i to data site d_j. Define stage out data matrix as:

$$
\begin{aligned}
M_{sod} &= \begin{pmatrix} SOD_{11} & SOD_{12} & \cdots & SOD_{1N_d} \\ SOD_{21} & SOD_{22} & \cdots & SOD_{2N_d} \\ \cdots & \cdots & \cdots & \cdots \\ SOD_{N_t1} & SOD_{N_t2} & \cdots & SOD_{N_tN_d} \end{pmatrix} \\
&= [SOD_{ij}]_{1\le i\le N_t, 1\le j\le N_d}
\end{aligned}
$$

Workflow scheduling is essentially a mapping from workflow W to the distributed systems S. Assign each task to an applicable computational site, and transfer data between computational sites and data sites accordingly. Figure 3.3 illustrates the mapping from W to S. t_1 and t_2 are assigned to p_1. Use $Ax(t_1) = p_1, Ax(t_2) = p_1$ to denote this assignment. Similarly, $Ax(t_3) = p_2, Ax(t_4) = p_3$. Define $EST(t_i, p)$ to be the earliest time that computational site p can execute task t_i.

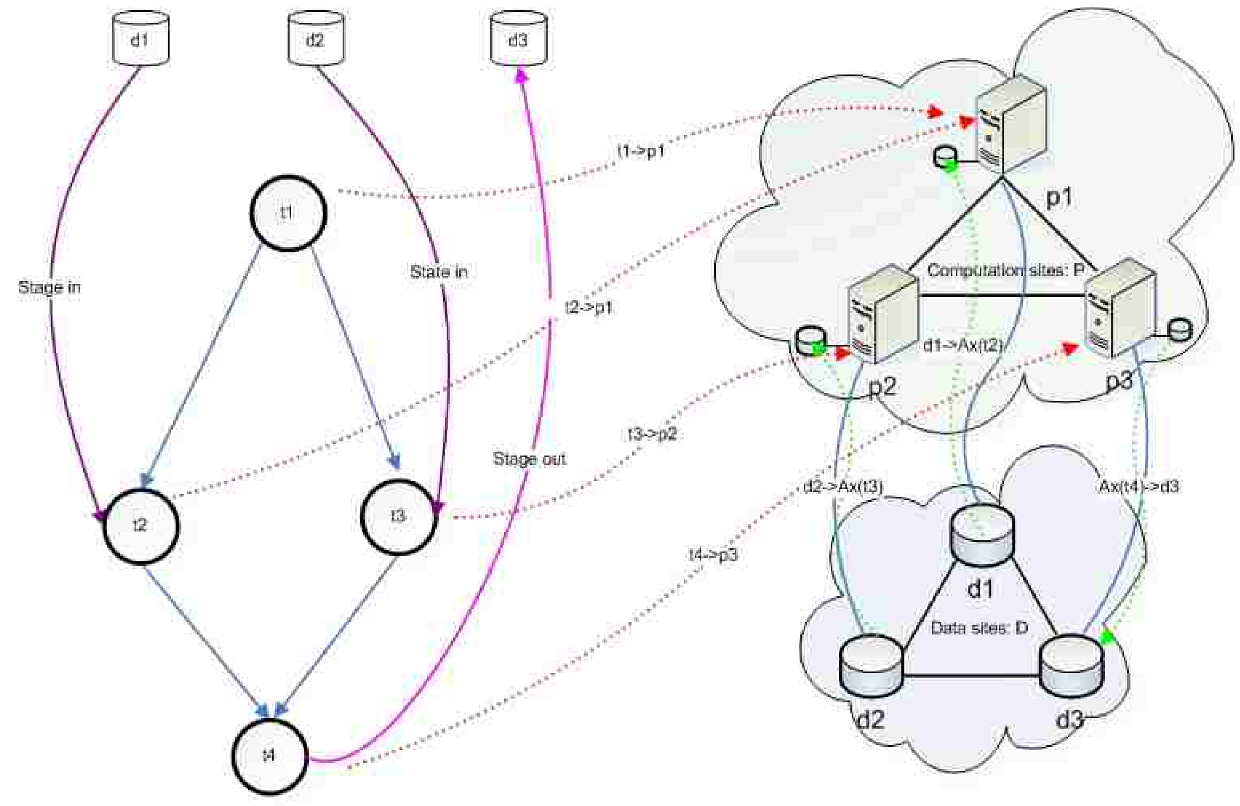


FIGURE 3.3: Mapping from DAG to sites

Define $WT(t_i, p)$ to be the time that task t_i have to wait on computational site p after p has finished the latest task. Define $SOT(t_i, p)$ to be the time at which the stage out process has done for task t_i on computational site p. Define $PT(p)$ to be the time at which tasks scheduled on computation site p has been done. Define $AS(W, p)$ to be a set of tasks that are assigned to computational sites p in the mapping process. Define the turn around time of the workflow W corresponding to a particular mapping m as $TR(W, m)$.Then we have:

$$PT(p_i) = max\left(\sum_{j\in AS(W,p_i)} (EXT(t_j, p_i) + WT(t_j, p_i)), MAX_{j\in AS(W,p_i)}(SOT(t_j, p_i))\right) \quad (3.1)$$

$$TR(W, m) = MAX_{i=1,2,\cdots,N_c}(PT(p_i)) \quad (3.2)$$

The objective function is:

$$MIN_{for\ all\ possible\ mapping\ m}(TR(W, m)) \quad (3.3)$$

3.6 Stage-in Data Placement Optimization

When there is a large data set in the data intensive workflow, there could be many mappings from data set to storage site. Picture 3.4 shows one possible mapping from tasks to computing site and

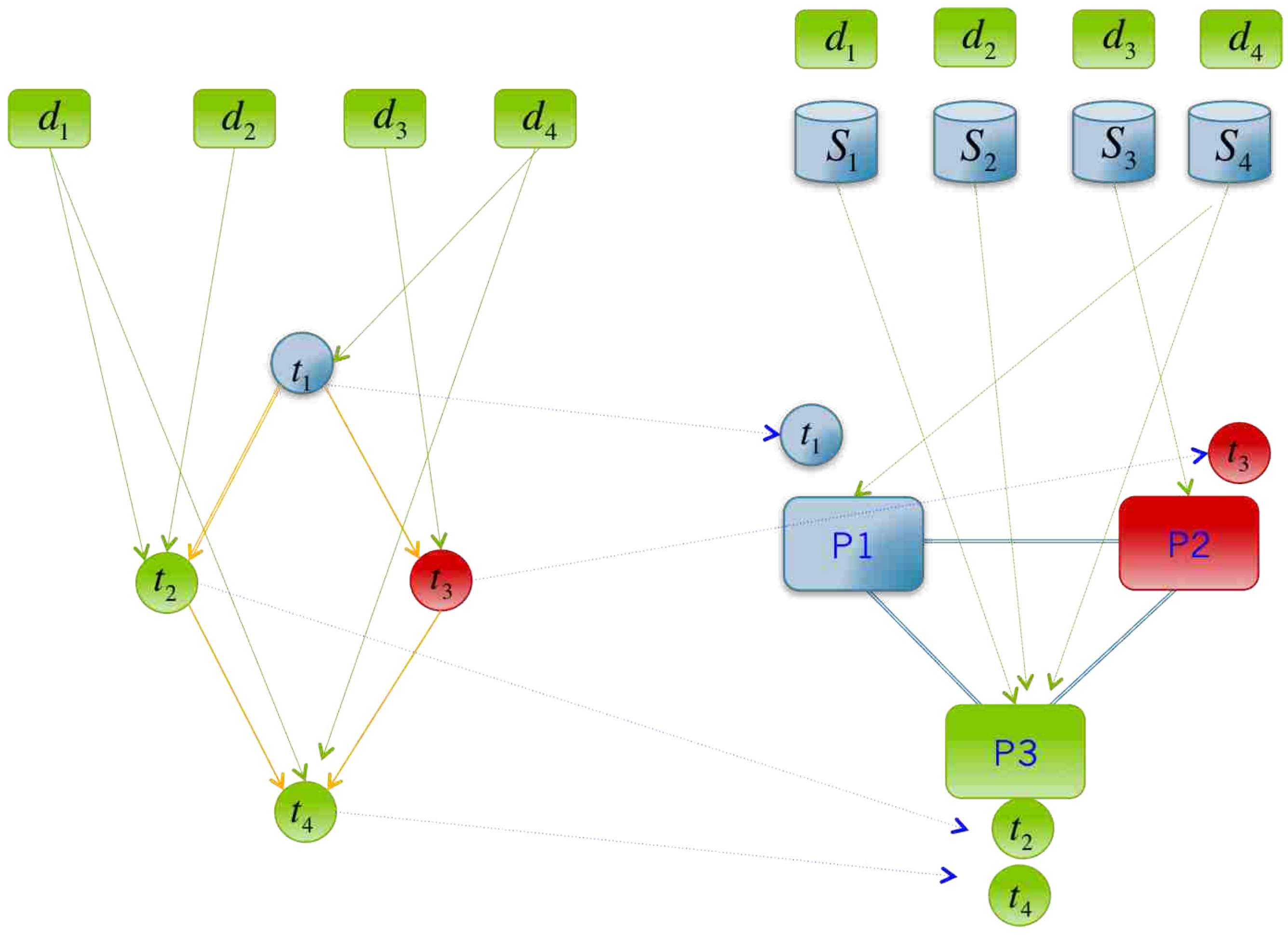


FIGURE 3.4: Mapping data files to storage sites

from stage in data to storage site. A poor task scheduling will increase the turnaround time of the workflow, resulting in an inefficient utilization of the computing resource. Similarly, a poor mapping from stage in site to storage site will degrade the overall performance since it will require a longer time to deliver the data to the computing site.

To formalize the stage in data location optimization problem, we assume there are a number of storage sties and computing sites connected by homogeneous or heterogeneous network as shown in Picture 3.5. There are many data files with different size to be delivered between the storage site and the computing site. Each file a fixed destination. The question is where to put each data file such that the deliver time for all the data files is minimized.

It is impossible to find an true optimal solution in polynomial time. In this research, a close to optimal algorithm is proposed. The main idea is to distribute data to each storage site proportional to the bandwidth between the storage site and the computing site. Make the data amount distributed

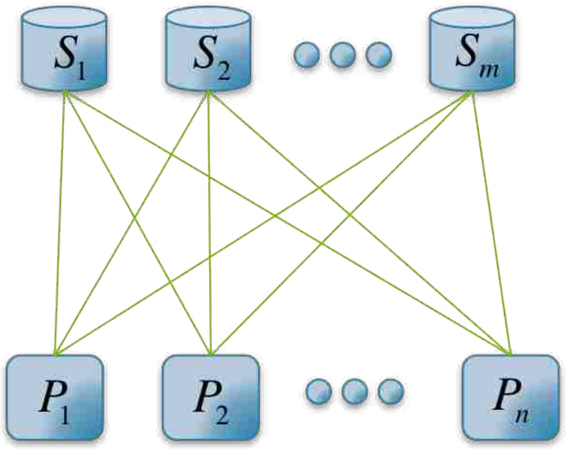


FIGURE 3.5: Formalizing the data placement optimization problem for data intensive workflow

to each storage site as close as possible to the desired portion. Make a replica of each data file when conflict occurs for placing a particular data file which is to be delivered to multiple computing site. In this case, this data file is the input for multiple tasks. Algorithm **??** describes the details to address the stage in data location optimization problem.

Algorithm 1 Stage in data location optimization

```
for all Computing site $P_i$ that has data to be staged in do
    Sort all the data files to be delivered to $P_i$ in descending order.
    Sort the bandwidth from all storage sites to $P_i$ in descending order.
    for all Storage site with connection to the computing site do
        First distribute data to the storage site with a higher bandwidth
        Distribute the larger data as early as possible.
        The data distributed to each storage site should be proportional to its bandwidth.
        Find the largest k such that the sum of the first k data are less than or equal to the desired
        data amount.
        Distribute them to the corresponding storage site.
        From the rest of the data, binary search the data closest to the difference between the
        portioning data and the sum of the first k data.
        Distribute the data to the corresponding site.
    end for
end for
```

The transfer time lower bound for the data to be delivered to a computing site is the quotient of the summation of the data amount and the summation of the bandwidth from each storage site to the computing site. The transfer time lower bound for all these computing sites is the maximum among them. Figure 3.6 and Figure 3.7 shows the experimental results for different number of data files.

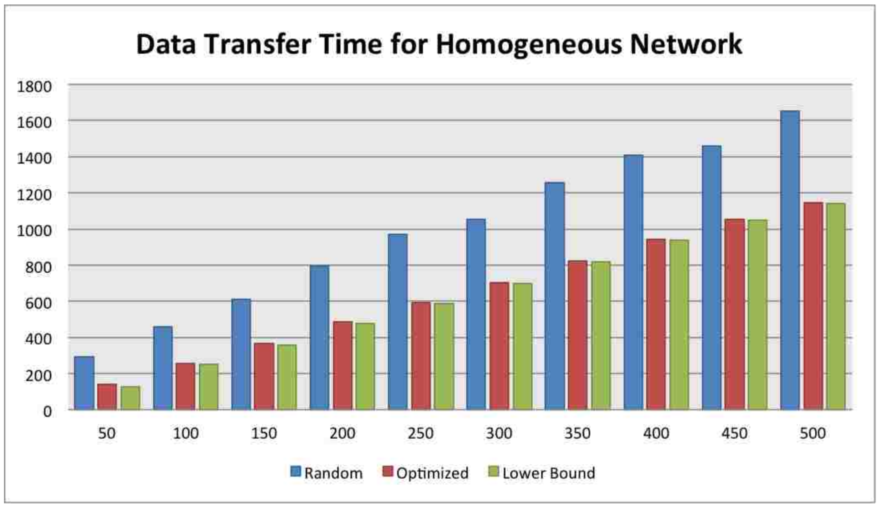


FIGURE 3.6: Optimizing the stage-in data location for homogeneous distributed systems

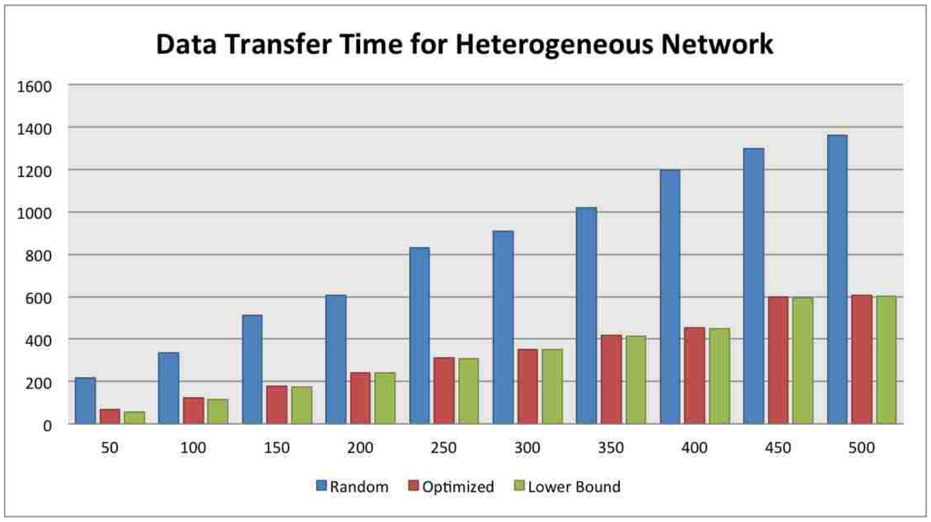


FIGURE 3.7: Optimizing the stage-in data location for heterogeneous distributed systems

The transfer time is surprisingly close to the lower bound for both homogeneous and heterogeneous distributed systems.

Chapter 4
Data-aware Optimal Scheduling

4.1 State Space Construction

The scheduling problem is essentially a state space search problem with precedence constraints. The constraints can be resolved by using topological sort with respect to the tasks. The order of task sequences obtained by applying topological sort is not unique. There might be many other complete task sequences in compliance with the topological sort. For instance, there are two legitimate task sequences for the task DAG in Figure 3.2, one of which is t_1, t_2, t_3, t_4 and the other is t_1, t_3, t_2, t_4. These legitimate task sequences form a large topological tree of the DAG. For instance, Figure 4.1 depicts the topological tree of this DAG. It can be observed that the depth of the topological tree is the number of the tasks of the DAG. The root of the tree is the first executable task and the leaf node is the last executable task.

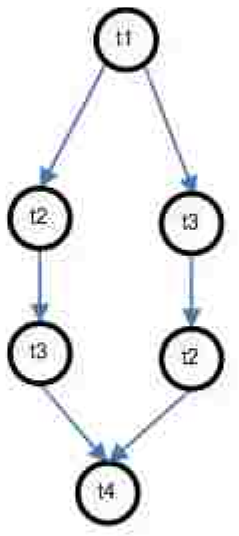


FIGURE 4.1: Topological tree

To construct a state space, first assign the root task to all the execution sites and check the legitimacy of the assignment. Only the legitimate one will be kept for further extension and the rest branches will be pruned. Each assignment from a task to an execution site corresponds to one node in the state space. For each node in the state space in appliance with the legitimacy constraints find the corresponding node in the topological tree, assign each child of the node in the topological tree to all computational sites, make all these assignments as child node of the node in the state space and check the legitimacy for further extension. Repeat the previous extension until the last node in the topological tree is assigned. An assignment is said to be legitimate if the following conditions are satisfied.

1. There exists a communication link between the assigned computational site of this task and the computational site where each task in its predecessor is assigned if it needs intermediate data from tasks belonging to its predecessor.

2. There exists a communication link between the assigned computational site of this task and the stage in data site if it needs extra data from remote sites.

3. There exists a communication link between the assigned computational site of this task and the stage out data site if it needs to store intermediate data to remote sites.

More Formally,

$$\begin{aligned} Ax(t) = p \textit{ is legitimate} \iff & \quad c_{Ax(t_i)p} \in C, \forall t_i \in Pre(t) \\ \wedge & \quad c_{d_ip} \in C, \forall d_i \in (ExPre(t) - Pre(t)) \\ \wedge & \quad c_{d_ip} \in C, \forall d_i \in (ExSuc(t) - Suc(t)) \end{aligned}$$

A state space for a workflow and distributed system in Figure 3.3 is shown in Figure 4.2.

4.2 State Space Search

Each path from the root to the target is a solution for the scheduling problem. The number of solutions is proportional to $N_t^{N_c}$, which makes it impossible to find the optimal or approximate optimal solution with a brute force approach such as breadth first search(BFS). A uniform cost search(UCS) approach maintains a priority queue which keeps the candidate states for expansion and the cost from the start state to the current state. Each time pop the states with the least cost and push the successors of that state into the priority queue. Continue this process until the goal state is reached. Although UCS avoids traverse the whole state space, it still needs to traverse upto an exponential magnitude of states.

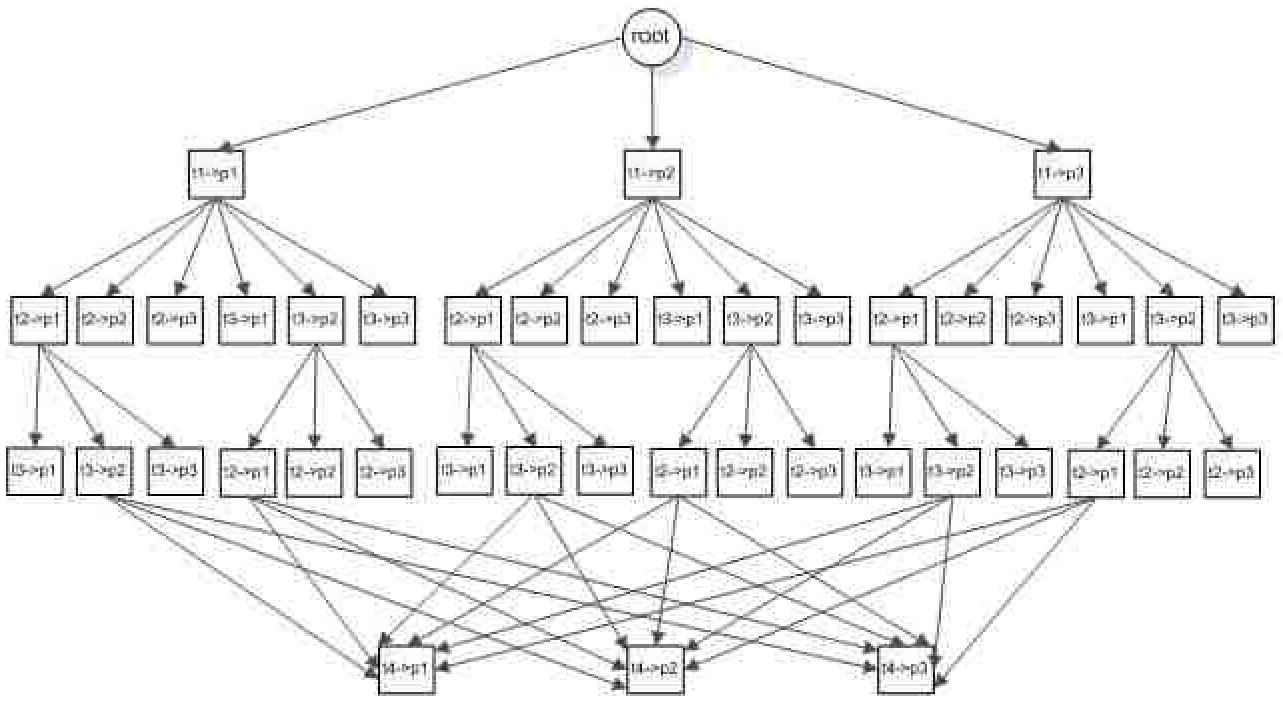


FIGURE 4.2: State space

A greedy search approach determines the next best state to visit by using a heuristic function which makes a best estimation of the cost from the current state to the goal state. For each successor of the current state, the cost is evaluated by the heuristic function, and the one with the least cost is treated as the best candidate state. Continue this process until the goal state is selected as the best candidate state.

The uniform cost search approach is not time efficient though it guarantee an optimal solution. On the contrary, the greedy search approach is time efficient although it does not guarantee an optimal solution. The A-star approach combines the advantages of UCS and greedy search approaches together by taking account of both efficiency and accuracy.

4.3 A-star Search

Similar to uniform cost search, A-star algorithm also has a priority queue which keeps a state and cost pair. However, the calculation of the cost is different. In the uniform cost search approach, the cost is measured by the cost from the start state to the current state, while in the A-star approach, the cost is measured by the summation of the cost from the start state to the current state and the estimated cost from the successor of the current state to the goal state.

Formally, use $f(x)$ to denote the cost function of state x. Use $g(x)$ to denote the actual cost from the start state to the current state. Use $h(x)$ to denote the underestimation from the successor to the goal state and $h^*(x)$ to denote the corresponding actual cost. Then we have, $f(x) = g(x) + h(x)$. It is impossible to know exactly the value of $h^*(x)$, otherwise the optimal solution is strait forward and we do not need to search the state space. What can be done is to make a best estimation of $h^*(x)$. Let $err = h^*(x) - h(x)$, then the smaller err is, the faster to reach the goal state. The most important thing in the A-star algorithm is to calculate $g(x)$ and $h(x)$.

4.4 Overlap of Task Execution and Data Movement

The workflow scheduling problem has been studied by many researchers and scientists. Most of them have assumed that the execution and data placement can not be overlapped. Based on this assumption, if two tasks t_i and t_j are scheduled on the same processor p, t_j cannot start execution when t_i has finished the computational part of the task even if all the data needed by t_j has arrived at p. The execution of t_j has to be postponed until t_i has transferred all the intermediate data to the computational sites where its successors are executed. In this study, this constraint is removed.

Each processor p_i maintains a list of queues corresponding to the rest computational sites, namely $CQ_1, CQ_2, \cdots, CQ_{i-1}, CQ_{i+1}, \cdots, CQ_{N_c}$, and a list of queues corresponding to the data sites, namely $DQ_1, DQ_2, \cdots, DQ_{N_d}$. When t_i finishes the computational part, for any task $t_j \in Suc(t_i)$, insert the intermediate data transfer task to $CQ_{Ax(t_j)}$. Also for any $d \in (ExSuc(t_i) - Suc(t_i))$, insert the stage out task to DQ_d. The data placement tasks within the same queue should be done sequentially, while the tasks in different queues can be done in parallel.

Define $ICT(t_i, t_j, Ax(t_i), Ax(t_j))$ to be the intercommunication time for the intermediate data transfer when t_i is assigned to $Ax(t_i)$ and t_j is assigned to $Ax(t_j)$. Define $SIT(t_i, Ax(t_i), d)$ to be the time spent on data stage in when t_i is assigned to $Ax(t_i)$ and the data is stored in remote data site d. Define $SOT(t_i, Ax(t_i), d)$ to be the time spent on data stage out when t_i is assigned to $Ax(t_i)$ and the data is supposed to be stored in remote data site d. Define $IND(p_i)$ to be the index of p_i, i.e,

$IND(p_i) = i$. Clearly, we have the following equations.

$$ICT(t_i, t_j, Ax(t_i), Ax(t_j)) = \begin{cases} \frac{IMD_{ij}}{TH_{IND(Ax(t_i))IND(Ax(t_j))}} & Ax(t_i) \neq Ax(t_j) \\ 0 & otherwise \end{cases}$$

$$SIT(t_i, Ax(t_i), d_j) = \frac{SID_{jIND(Ax(t_i))}}{TH_{jIND(Ax(t_i))}}$$

$$SOT(t_i, Ax(t_i), d_j) = \frac{SOD_{IND(Ax(t_i))j}}{TH_{IND(Ax(t_i))j}}$$

Use Ax to denote the partial assignment from the start state to the current state x in the state space. $PT(p_i)$ is the already known definitely processing time of processor p_i with respect to Ax. $DT(d_i)$ is the already known definitely data transfer time of remote data site d_i. A_{p_i} is the set of tasks that are assigned to processor p_i with respect to Ax. Define SP_{t_i} to be a set of processors that consist of the processors to which the predecessors of t_i are assigned, excluded the one to which t_i is assigned.

$$SP_{t_i} = \{Ax(t) : \forall t \in Pre(t_i)\ Ax(t) \neq Ax(t_i)\} \tag{4.1}$$

Define R_{p_i} to be a set of tasks that are assigned to a processor $p_i \in SP_{t_i}$ and will transfer intermediate data to $Ax(t_i)$. Define $S_{Ax(t_i)}$ to be a set of tasks that consumes the intermediate data from tasks in R_{p_i}.

$$R_{p_i} = \{t_{\alpha_1}, t_{\alpha_2}, \cdots, t_{\alpha_m} : t_{\alpha_1}, t_{\alpha_2}, \cdots, t_{\alpha_n}\ are\ in\ the\ order\ of\ appearance\ in\ A_x\} \tag{4.2}$$

$$S_{Ax(t_i)} = \{t_{\beta_1}, t_{\beta_2}, \cdots, t_{\beta_m} : t_{\beta_i}\ corresponds\ to\ t_{\alpha_i}, 1 \leq i \leq m\} \tag{4.3}$$

Define $ETT(t_{\alpha_i})$ to be the earliest time to transfer data produced by t_{α_i}.

$$ETT(t_{\alpha_i}) = \begin{cases} EST(t_{\alpha_i}, Ax(t_{\alpha_i})) + EXT(t_{\alpha_i}, Ax(t_{\alpha_i})) & i = 1 \\ max(EST(t_{\alpha_i}, Ax(t_{\alpha_i})) + EXT(t_{\alpha_i}, Ax(t_{\alpha_i})), & \\ ETT(t_{\alpha_{i-1}}) + ICM(t_{\alpha_{i-1}}, t_{\beta_{i-1}}, Ax(t_{\alpha_{i-1}}), Ax(t_{\beta_{i-1}}))) & 1 < i \leq m \end{cases} \tag{4.4}$$

Define SI_{t_i} to be a set of data sites from which data will be staged in to the computational sites $Ax(t_i)$ where t_i is scheduled.

$$SI_{t_i} = \{I_1, I_2, \cdots, I_{N_{si}}\} \tag{4.5}$$

Define Γ_{SI_k} to be a set of tasks which need data to be staged in from remote site I_k to $Ax(t_i)$.

$$\Gamma_{SI_k} = \{t_{\gamma_1}, t_{\gamma_2}, \cdots, t_{\gamma_m} : t_{\gamma_1}, t_{\gamma_2}, \cdots, t_{\gamma_m}\ are\ in\ the\ order\ of\ appearance\ in\ A_x\} \tag{4.6}$$

Define $ESI(t_{\gamma_i}, I_j)$ to be the earliest time to stage in the data of t_{γ_i} from I_j.

$$ESI(t_{\gamma_i}, I_j) = \begin{cases} 0 & i = 0 \\ \sum_{i=0}^{i-1} SIT(t_{\gamma_i}, Ax(t_{\gamma_i}), I_j) & i > 0 \end{cases} \tag{4.7}$$

Define SO_{t_i} to be a set of data sites to which data will be staged out from the computational sites $Ax(t_i)$ where t_i is scheduled.

$$SO_{t_i} = \{O_1, O_2, \cdots, O_{N_{so}}\} \tag{4.8}$$

Define Δ_{SO_k} to be a set of tasks which generate intermediate data to be staged out to the remote site O_k from $Ax(t_i)$.

$$\Delta_{SO_k} = \{t_{\delta_1}, t_{\delta_2}, \cdots, t_{\delta_m} : t_{\delta_1}, t_{\delta_2}, \cdots, t_{\delta_m} \; are \; in \; the \; order \; of \; appearance \; in \; A_x\} \tag{4.9}$$

Define $ESO(t_{\delta_i}, O_j)$ to be the earliest time to stage out the data of t_{δ_i} to O_j.

$$ESO(t_{\delta_i}, O_j) = \begin{cases} EST(t_{\delta_i}, Ax(t_{\delta_i})) + EXT(t_{\delta_i}, Ax(t_{\delta_i})) & i = 1 \\ max(ESO(\delta_{i-1}, O_j) + SOT(t_{\delta_{i-1}}, Ax(t_{\delta_{i-1}}), O_j), & \\ EST(t_{\delta_i}, Ax(t_{\delta_i})) + EXT(t_{\delta_i}, Ax(t_{\delta_i}))) & i > 1 \end{cases} \tag{4.10}$$

Define $\alpha(.)$ to be a function mapping a task t_i to a task in R_{p_k}.

$$\alpha(t_i) = t_{\alpha_j} \Leftrightarrow t_i = t_{\alpha_j} \tag{4.11}$$

Define $\gamma(.)$ to be a function mapping a task t_i to a task in Γ_{SI_k}.

$$\gamma(t_i) = t_{\gamma_j} \Leftrightarrow t_i = t_{\gamma_j} \tag{4.12}$$

Define $\delta(.)$ to be a function mapping a task t_i to a task in Δ_{SO_k}.

$$\delta(t_i) = t_{\delta_j} \Leftrightarrow t_i = t_{\delta_j} \tag{4.13}$$

We can derive the following equation.

$$EST(t_i, Ax(t_i)) = max \begin{pmatrix} \underset{\forall I_x \in SI_{t_i}}{\text{MAX}} (ESI(\gamma(t_i), I_x) + SIT(t_i, Ax(t_i), I_x)), \\ \underset{\forall t_j \in Pre(t_i)}{\text{MAX}} (ETT(\alpha(t_j)) + ICT(t_j, t_i, Ax(t_j), Ax(t_i))) \end{pmatrix} \tag{4.14}$$

$$PT(p_i) = max\left(\begin{array}{c}\underset{\forall t_j \in A_{p_i}}{\text{MAX}}\underset{\forall O_j \in SO_{t_j}}{\text{MAX}}(ESO(\gamma(t_j), O_j) + SOT(t_j, Ax(t_j), O_j)),\\ \underset{\forall t_j \in A_{p_i}}{MAX}(EST(t_j, Ax(t_j)) + EXT(t_j, Ax(t_j)))\end{array}\right) \tag{4.15}$$

Define $URT(t_i)$ to be an under estimated remaining execution time from a child of the current task t_i to the end of the workflow.

$$URT(t_i) = \begin{cases} 0 & if\ t_i\ is\ the\ last\ task \\ \underset{\forall t_j \in Suc(t_i)}{\text{MAX}}(\underset{\forall p_k \in P}{\text{MIN}}(EXT(t_j, p_k) + URT(t_j))) & otherwise \end{cases} \tag{4.16}$$

Define $UTT(t_j, Ax)$ to be the underestimated turnaround time corresponding to a partial assignment Ax where tasks from the root to t_j has been scheduled and the execution time of the rest tasks are underestimated.

$$UTT(t_j, Ax) = max\left(\begin{array}{l}\underset{\forall t_j \in A_{p_i}}{MAX}(EST(t_j, Ax(t_j)) + EXT(t_j, Ax(t_j) + \underset{\forall t_k \in Suc(t_j) \wedge t_k \notin A_p, \forall p \in P}{\text{M}IN}\\ (EXT(t_k, p) + ICT(t_j, t_k, Ax(t_j), Ax(t_k)) + URT(t_k))),\\ \underset{\forall p_i \in P}{\text{M}AX}PT(p_i)\end{array}\right) \tag{4.17}$$

4.5 The Algorithm for Co-scheduling

In order to find an optimal task mapping, a naive way is to compare the turnaround time of all the possible mappings and choose the one leading to the shortest turnaround time. The naive approach will definitely find the optimal solution, however, the computation complexity could be of unpreventably high magnitude, which makes it impossible to apply the brute-force technique when the workflow or the distributed system is in large scale.

Each mapping is essentially a process of assigning each task of the workflow to a appropriate computational site, step by step, in compliance with the dependency constraints. It is a full assignment from the first step to the last step. At each intermediate step, it is a partial assignment. At each step, we can estimate an upper bound of the turnaround time for the workflow. This estimation consists of the actual time cost for the tasks from the first assignment to the current step and an underestimate time cost from the current step to the last step. Considering all the possible partial assignments, at each step, determine the one with the lowest upper bound, which is considered to be on the potentially optimal mapping path with high probability. Then we will move forward one step further from that step by mapping all the ready tasks to all the possible computational sites. Update the underestimated turnaround time and determine the new state with a lowest upper bound. Repeat this process until a full assignment for a particular mapping is reached and the turnaround time for this full assignment is less than or equal to the underestimate turnaround time at the current step of any partial assignment for all the rest mappings. The detail of the algorithm is shown in Algorithm 2.

Algorithm 2 Heuristic Search

```
1.  initialize a root node for state space search tree without any task assignment
2.  set current node $curNode$ to be the root node
3.  calculate the set of ready for execution tasks $S_{ready} = CalReadySet(root, curNode)$
4.  while $S_{ready} \neq \emptyset$ do
5.      for all $t_i \in S_{ready}$ do
6.          for all $p_j \in P$ do
7.              if the assignment of $t_i$ to $p_j$ is legitimated then
8.                  expand the state space search tree by generating a new node $N_{ij} = (t_i, p_j)$
9.                  set $N_{ij}$ as the child of the current node
10.                 calculate $UTT_{ij}$ of $N_{ij}$ and insert $UTT_{ij}$ into a queue
11.             end if
12.         end for
13.     end for
14.     extract the element with the smallest $UTT$ from the queue
15.     identify the corresponding node of the state space search tree
16.     update the current node $curNode$ as the corresponding node
17.     calculate the set of ready for execution tasks $S_{ready} = CalReadySet(root, curNode)$
18. end while
```

To better illustrative the algorithm, we describe a simple example in the following. The workflow and distributed system are the same as which are described in Figure 3.3. The task computation time matrix, intermediate data matrix, data stage in and stage out matrix are also given as following.

$$M_c = \begin{pmatrix} 10 & 200 & 400 \\ 20 & 300 & 500 \\ 300 & 10 & 400 \\ 300 & 500 & 10 \end{pmatrix}$$

$$M_{imd} = \begin{pmatrix} 0 & 10 & 10 & 0 \\ 0 & 0 & 0 & 20 \\ 0 & 0 & 0 & 10 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$M_{sid} = \begin{pmatrix} 0 & 20 & 0 & 0 \\ 0 & 0 & 20 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$M_{sid} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 10 \end{pmatrix}$$

Assume the throughput of the links for computational sites to computational sites, computational sites to data sites are unit one. Then we can derive the following state space search tree (Figure 4.3).

4.6 Data-aware Algorithm Evaluation

The evaluation of the proposed algorithm consists of two scenarios. In scenario one, we assume that there is no stage in and stage out data. There are only intermediate data between tasks. In scenario two, we assume that there are some tasks need data to be staged in before execution, and that some tasks need data to be staged out after execution.

4.6.1 Overlap of Execution and Data Transfer

In [46], the authors assume that the task execution process is immediately started after all the data from the dependent tasks, and that during the data transfer process, both the sending and receiving sites should be free from execution any other tasks. In [30], the author assumes that it is not necessarily to start the execution of a task immediately even though the data are ready. Contrarily, the execution of the ready task can be postponed until some data required by other tasks are transferred.

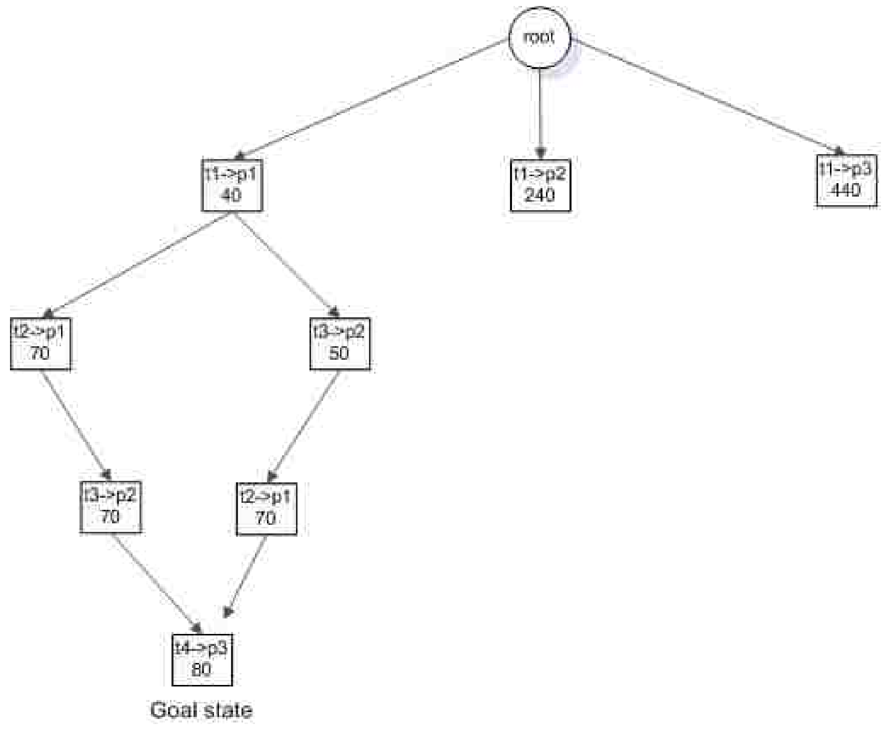


FIGURE 4.3: State space search tree

Increasing the flexibility of data transfer orders will shorten the turnaround time of the workflow in some cases. However, the computation complexity will increase dramatically since it will consider all the possible data transfer and task execution orders for the tasks assigned to one particular computational site.

Even though the resulting scheduling is optimal under their particular assumptions, neither of these assumptions are ideal. We aim to find a way to minimize the turnaround time as much as possible, meanwhile not to increase the time complexity. It is possible to achieve this goal by allowing overlap of the execution of a task and data transfer of another independent task. Under this assumption, we can achieve a comparable performance with the one in [30] and a similar time complexity with that in [46].

This example is the same to that in [46] and [30]. Figure 4.4 shows the workflow and the distributed system. The number besides the edge of the workflow represents the intermediate data amount between dependent tasks. M_c is the task computation time matrix. The throughput of each network link shown in the distributed system is one.

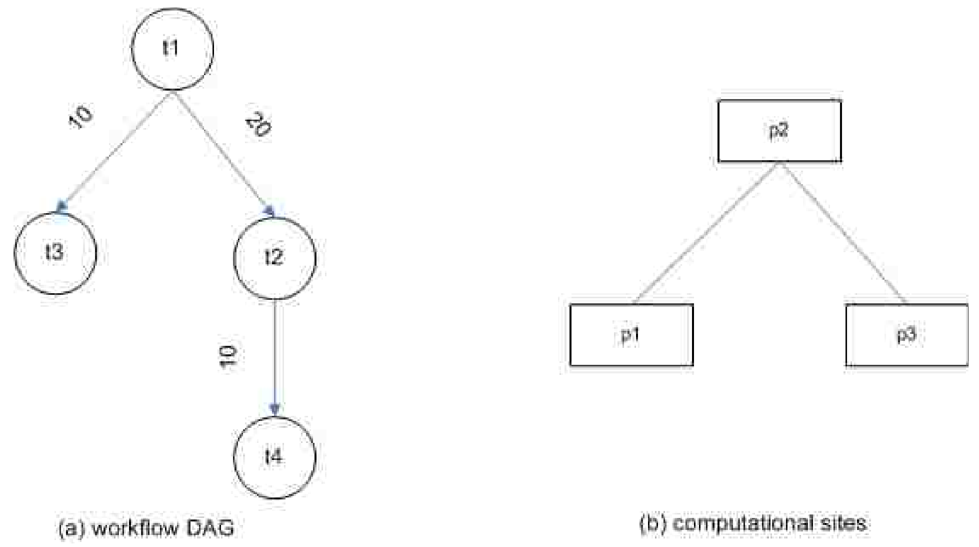


FIGURE 4.4: Workflow DAG and computational sites

$$M_c = \begin{pmatrix} 300 & 50 & 400 \\ 60 & 400 & 600 \\ 70 & 500 & 700 \\ 800 & 80 & 900 \end{pmatrix}$$

Figure 4.5 illustrates the state space search tree in the process of solving the problem under our assumption. Figure 4.6 compares the optimal turnaround time under different assumptions. Figure 4.6(a)-(c) represents the results derived from assumptions of [46], [30] and ours. In Figure 4.6(c), the dotted line represents the data transfers and the solid line represents the task executions. The dotted line is placed parallel above the solid line, meaning that the data transfer can be overlapped with the task execution. As we expected, the turnaround time will be much shorter if we allow overlap of data transfer and task execution as shown in Figure 4.6.

4.6.2 Taking Account of Stage-in and Stage-out Data

Most of the works done so far consider only the intermediate data transfers with the aim to minimize the turnaround time of the workflow. However, in the reality, some tasks might need extra data to be staged in from remote data sites as input before execution. Similarly, some tasks might need to stage out some intermediate data, which might be valuable to some users, to the remote sites. The optimal

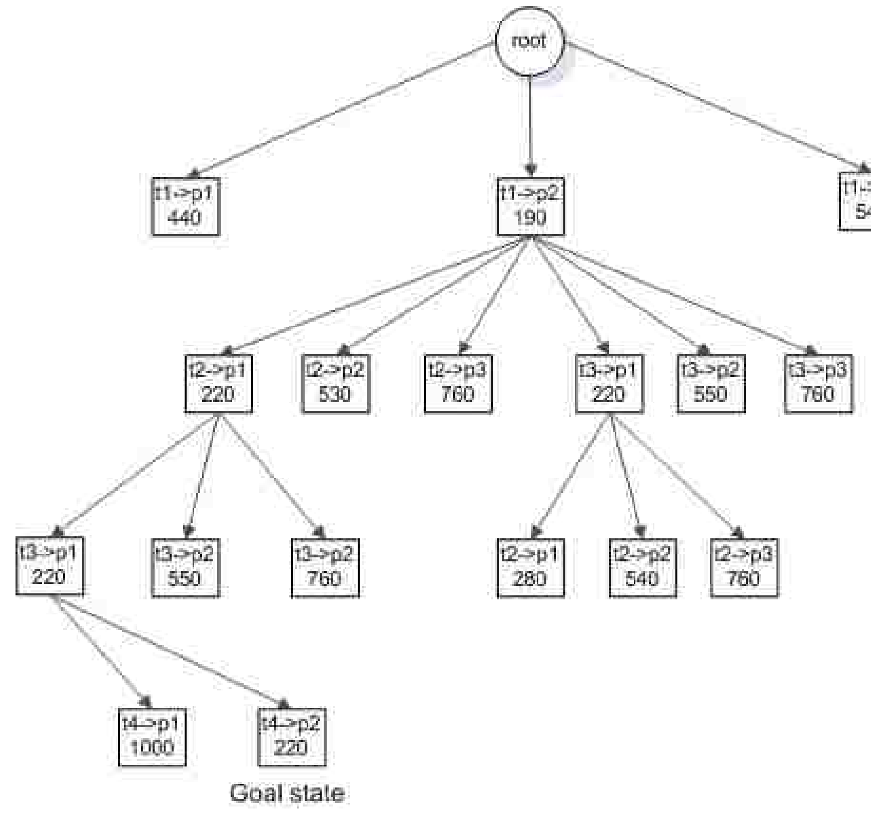
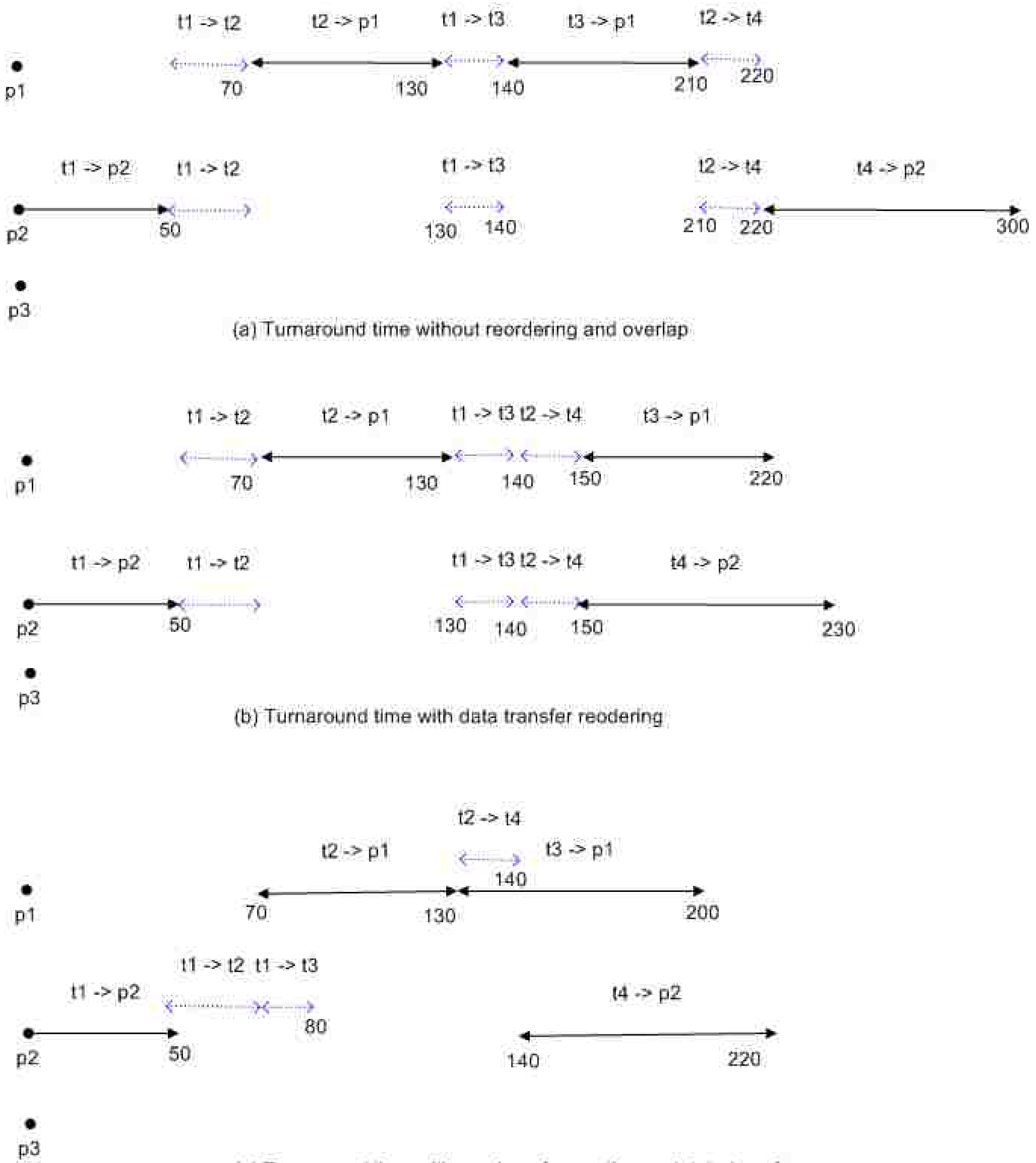


FIGURE 4.5: State space search tree

scheduling derived without considering the stage in and stage out is not always optimal when they are considered. In the following, we will give an example to verify this statement.

Figure 4.7 depicts the workflow and the distributed system, which consists of heterogeneous interconnected computational sites and data sites. The numbers appear on the edge of the workflow reflect the data amount to be transferred and the numbers appear on the link of the sites represent the estimated throughput of that link. The task execution time matrix is represented by M_c in the following.

$$M_c = \begin{pmatrix} 10 & 100 & 200 \\ 10 & 20 & 400 \\ 200 & 10 & 300 \\ 400 & 10 & 20 \end{pmatrix}$$



(a) Turnaround time without reordering and overlap

(b) Turnaround time with data transfer reodering

(c) Turnaround time with overlap of execution and data transfer

FIGURE 4.6: Turnaround time of workflow

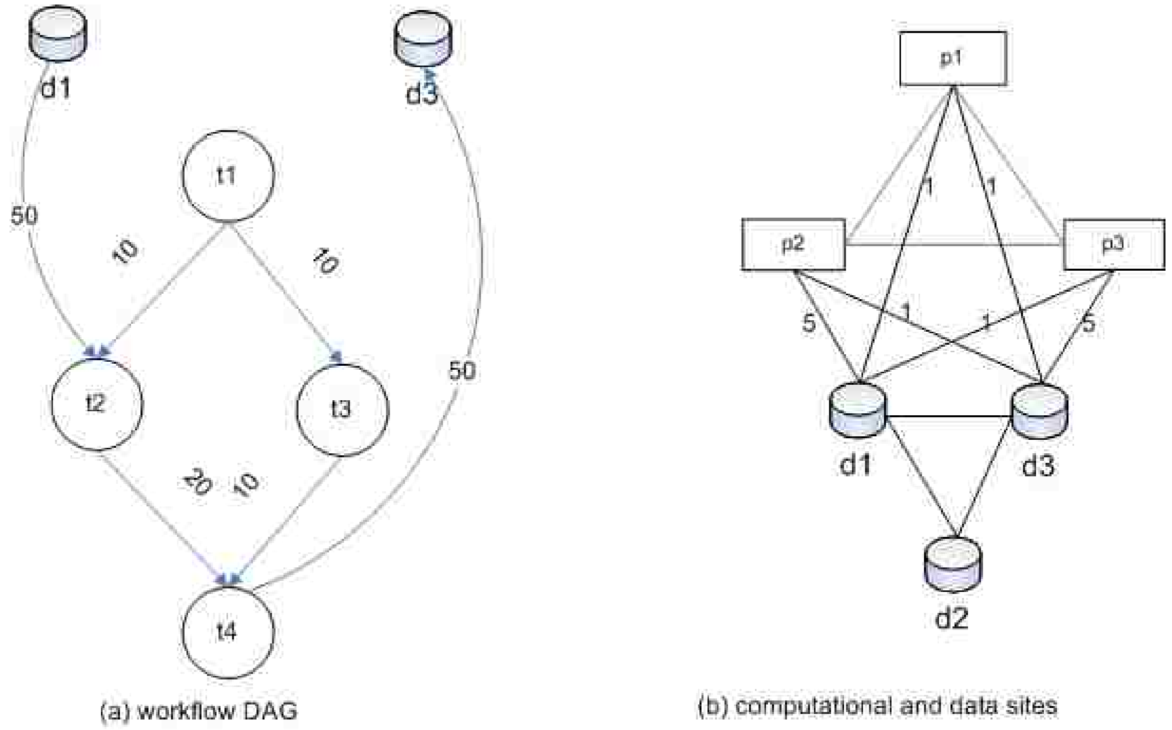


FIGURE 4.7: Workflow DAG and sites

Now let's compare two scheduling schemes, namely scheduling 1 and scheduling 2. For scheduling 1, tasks t_1 and t_2 are mapped to computational site p_1, tasks t_3 and t_4 are mapped to computational site $p2$. For scheduling 2, task t_1 is mapped on computational site p_1, tasks t_2 and t_3 are mapped to computational site p_2, task t_4 is mapped to computational site p_3.

Suppose we have another workflow which is exactly the same as described in Figure 4.7 except that there is no data stage in and stage out, then we can derive that scheduling 1 is the optimal solution. For the workflow described in Figure 4.7 we can derive that scheduling 2 is the optimal solution when we consider the data stage in and stage out. Figure 4.8 compares the turnaround time for scheduling 1 and scheduling 2 without considering the stage in and stage out. It turns out the scheduling 1 is better than scheduling 2. Figure 4.9 compares the turnaround time for scheduling 1 and scheduling 2 when considering the stage in and stage out. It turns out the scheduling 2 is better than scheduling 1. Most of the current works neglect the importance of taking account of the stage in and stage out data, hence they will lead to scheduling 1 for the workflow described in Figure 4.7, which is not optimal. However, with our approach, we can derive an optimal solution for the workflow described in Figure 4.7 as scheduling 2.

TABLE 4.1: Two scheduling schemes comparison

scheduling 1	$t_1 \rightarrow p_1$	$t_2 \rightarrow p_1$	$t_3 \rightarrow p_2$	$t_4 \rightarrow p_2$
scheduling 2	$t_1 \rightarrow p_1$	$t_2 \rightarrow p_2$	$t_3 \rightarrow p_2$	$t_4 \rightarrow p_3$

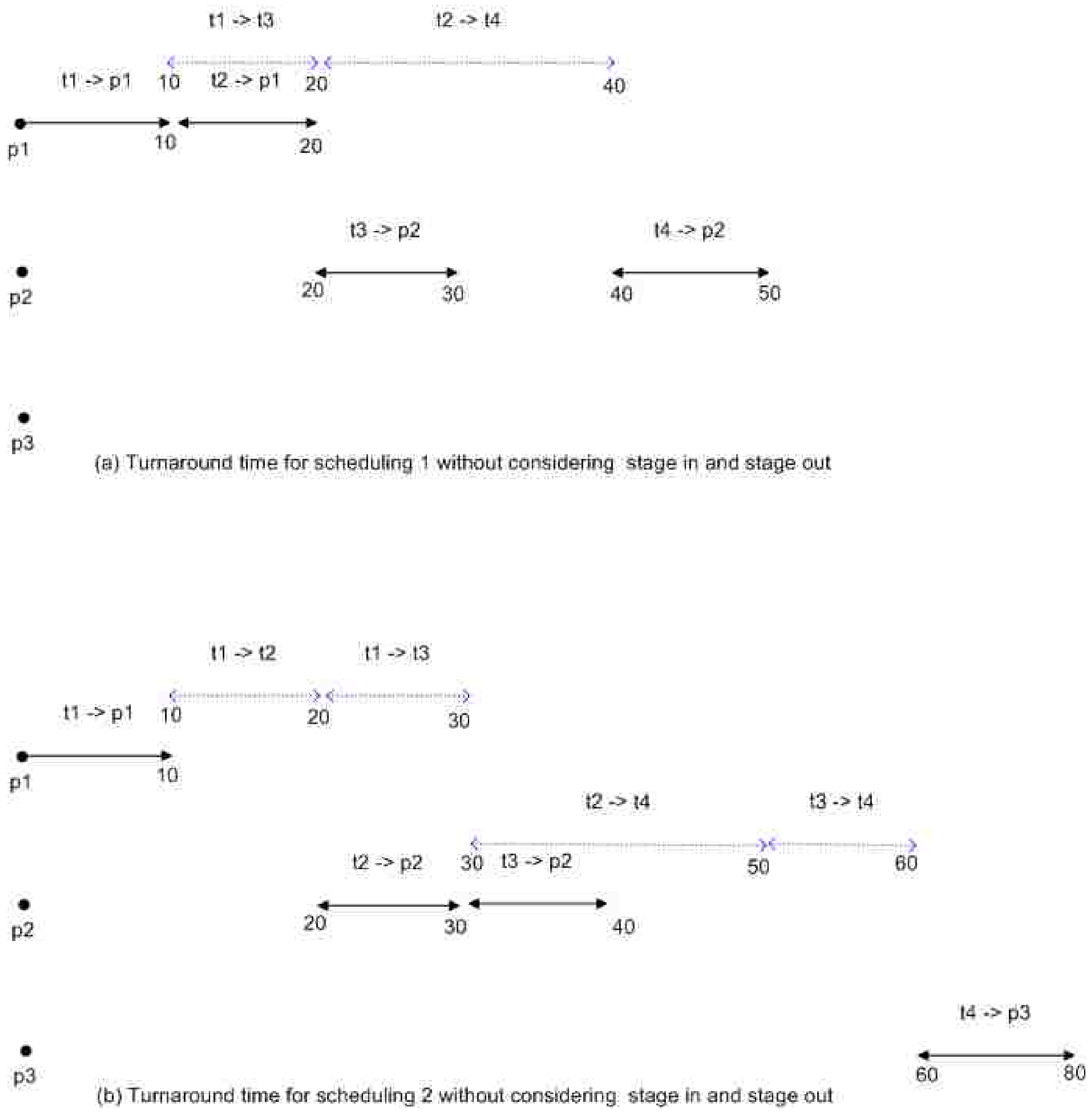
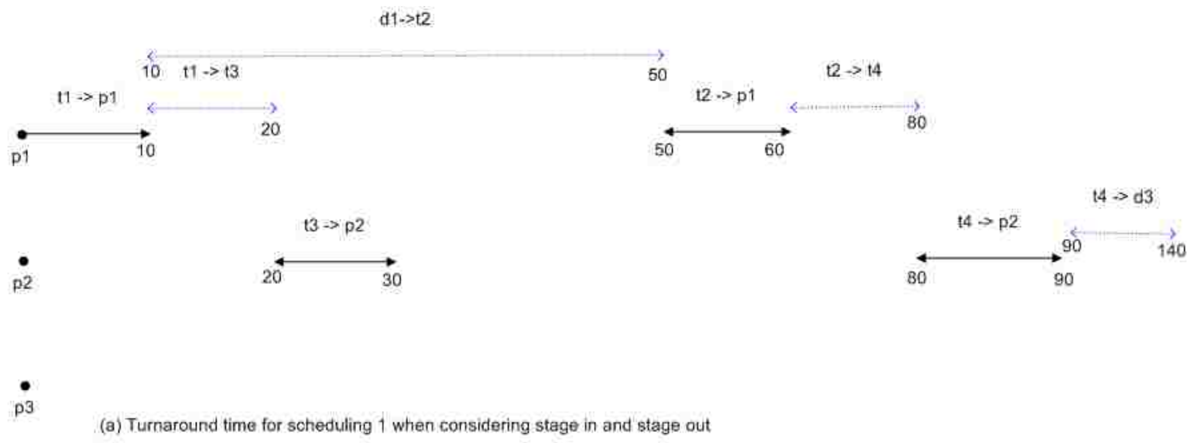


FIGURE 4.8: Turnaround time without co-scheduling



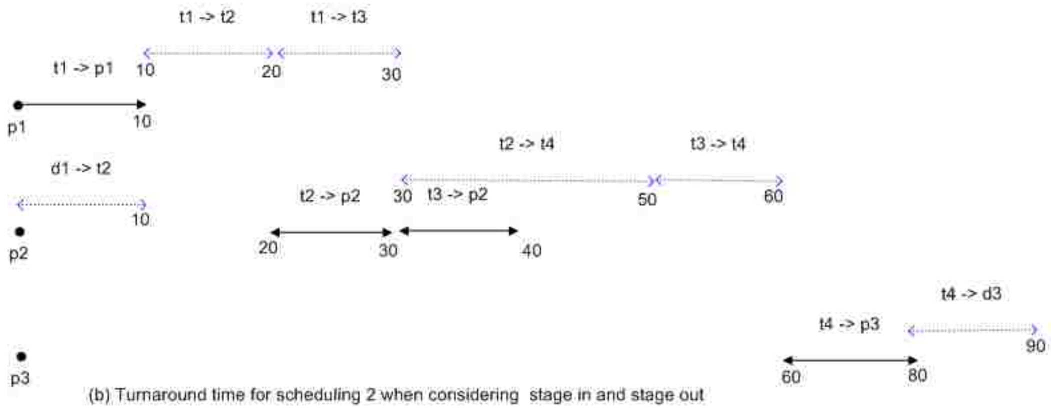


FIGURE 4.9: Turnaround time with co-scheduling

TABLE 4.2: Terms defined for problem input

Terms	Definition
$P = \{p_i\}$	A set of computational sites
$D = \{d_i\}$	A set of data sites
$C = \{c_{p_i p_j}, c_{d_k d_l}, c_{p_i d_k}\}$	A set of links between computational sites and data sites
N_c	Number of computation sites
N_d	Number of data sites
EXT_{ij}	The execution time of t_i on p_j
TH_{ij}	The throughput between site i and site j
IMD_{ij}	The amount of intermediate data between t_i and t_j
SID_{ij}	The amount of stage in data from data site d_i to task t_j
SOD_{ij}	The amount of stage out data from task t_i to data site d_j
M_c	The task computation matrix
M_t	The throughput matrix
M_{imd}	The intermediate data matrix
M_{sid}	The stage in data matrix
M_{sod}	The stage out data matrix

TABLE 4.3: Terms defined during calculation

Terms	Definition
$Pre(t_i)$	A set of immediate predecessors of t_i in a conventional workflow
$ExPre(t_i)$	A set of immediate predecessors of t_i in an extended workflow
$Suc(t_i)$	A set of immediate successors of t_i in a conventional workflow
$ExSuc(t_i)$	A set of immediate successors of t_i in an extended workflow
$Ax(t_i)$	The computation site which t_i is assigned to
$EXT(t_i, p_j)$	The execution time when t_i is assigned to p_j
$ICT(t_i, Ax(t_i), t_j, Ax(t_j))$	The intermediate data communication time
$SIT(t_i, Ax(t_i), d_j)$	The stage in time from d_j to $Ax(t_i)$
$SOT(t_i, Ax(t_i), d_j)$	The stage out time from $Ax(t_i)$ to d_j
$EST(t_i, p_j)$	The earliest time that t_i can start execution on p_j
$ETT(t_{\alpha_i})$	The earliest time to transfer the data generate by t_{α_i}
$ESI(t_{\gamma_i}, I_j)$	The earliest time to stage in the data needed by t_{γ_i} from data site I_j

4.7 Experimental Results

In order to examine the benefits of applying overlap scheme in the data-aware scheduling algorithms, we compare both the turnaround time and the number of nodes expanded during the state space searching process. In this simulation, we choose the DAG structure similar to the one in Figure 4.10(a) except that there is no stage in and stage out data in this simulation. We set up the computation time for each task on each computational site as a uniform distribution. The mean is 720 and the upper bound is 10% higher and the lower bound is 10% lower. The intermediate data between tasks are also uniform distribution. We consider 5 cases, the mean for these 5 case are separately 1/2, 1/3, $\cdots$, 1/6 of 720, and the upper bound is 10% higher and the lower bound is 10% lower. Figure 4.12(a) reflects the turnaround time of these 5 cases. It shows that the turnaround time can be decreased around 20 percents when the intermediate communication time is around 1/2 of the computation time.

Figure 4.12(b) depicts the number of nodes expanded during the state space search process. Since the algorithm proposed by Lin [30] extended the workflow by adding a communication node between every tow tasks, the problem size will be doubled. Also during the state space search process, all possible orders should be considered, hence the state space will be very huge. The result shows that our design is quite efficient in time.

In order to verify the benefit of co-scheduling scheme in the data-aware scheduling algorithm, we choose a DAG with the same structure to the one in Figure 4.12(c). The computation time for each task and the intermediate data amount are both uniformly distributed with a mean of 720. We vary the stage in and stage out data amount with a mean of 1, 2, $\cdots$, 5 times of 360. First we make the network bandwidth between the stage in/out data site and the computational data site as homogeneous. Figure 4.12(d) shows the result. It seems that there are some benefit by using co-scheduling algorithms, however, not significant. Then we set up the network bandwidth between the stage in/out data site and the computational data site as heterogeneous, the turnaround time will be significantly decreased when using co-scheduling algorithm.

We also vary the structure of the workflow and do a similar simulation as mentioned above. The workflows we have considered are separately linear-structured, merging-structured, emission-structured and merging-emission structured as shown in Figure 4.11. The turnaround time by using our data-aware scheduling algorithm is much better than Lin's optimal scheduling algorithm when scheduling a workflow with stage in/out data to heterogeneous distributed systems.

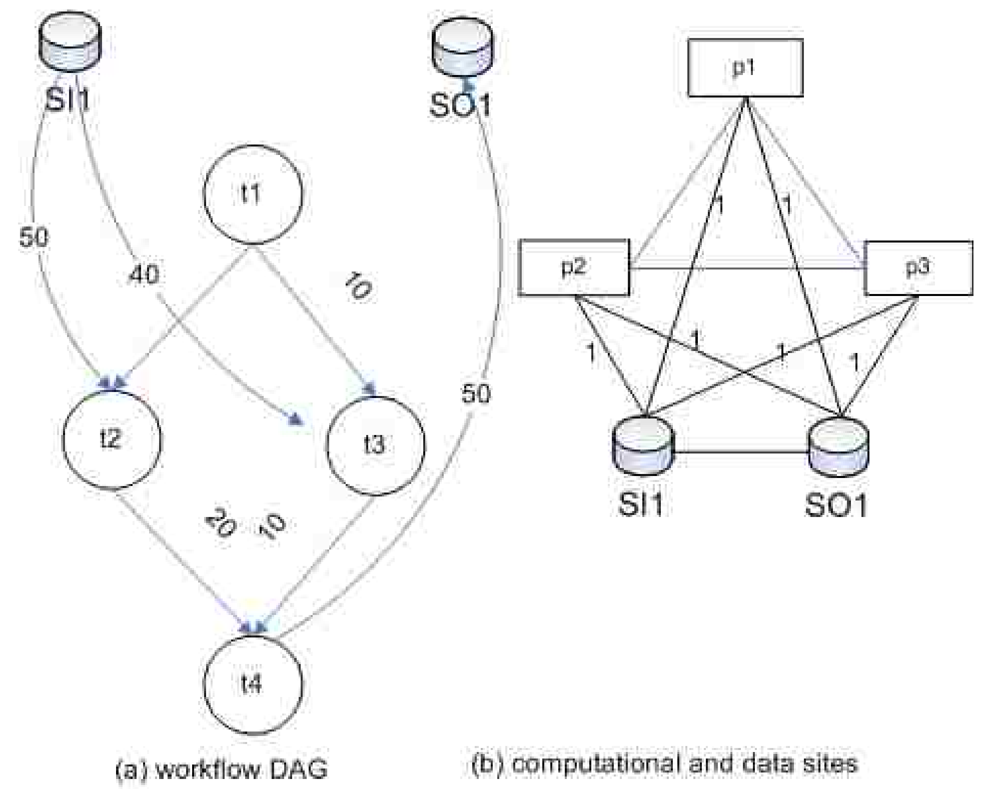


FIGURE 4.10: The workflow and the distributed systems

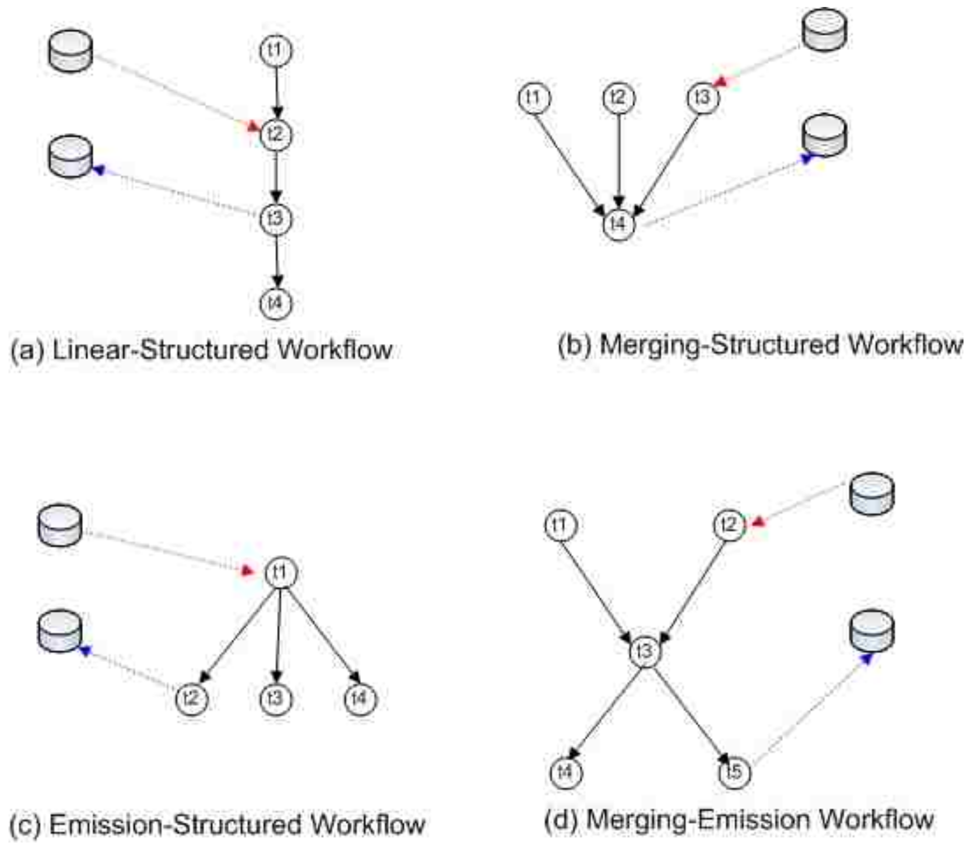

t1
t2
t3
t4
(a) Linear-Structured Workflow
t1
t2
t3
t4
(b) Merging-Structured Workflow
t1
t2
t3
t4
(c) Emission-Structured Workflow
t1
t2
t3
t4
t5
(d) Merging-Emission Workflow

FIGURE 4.11: Typical workflow structures

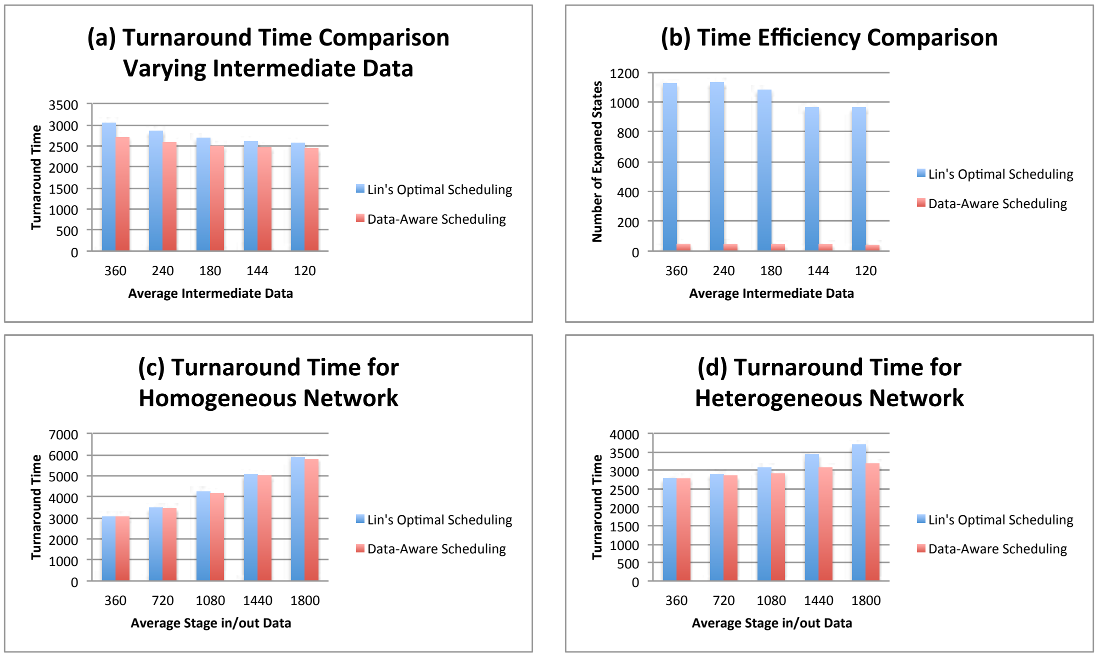


FIGURE 4.12: Comparison of Data-aware scheduling and Lin's optimal scheduling

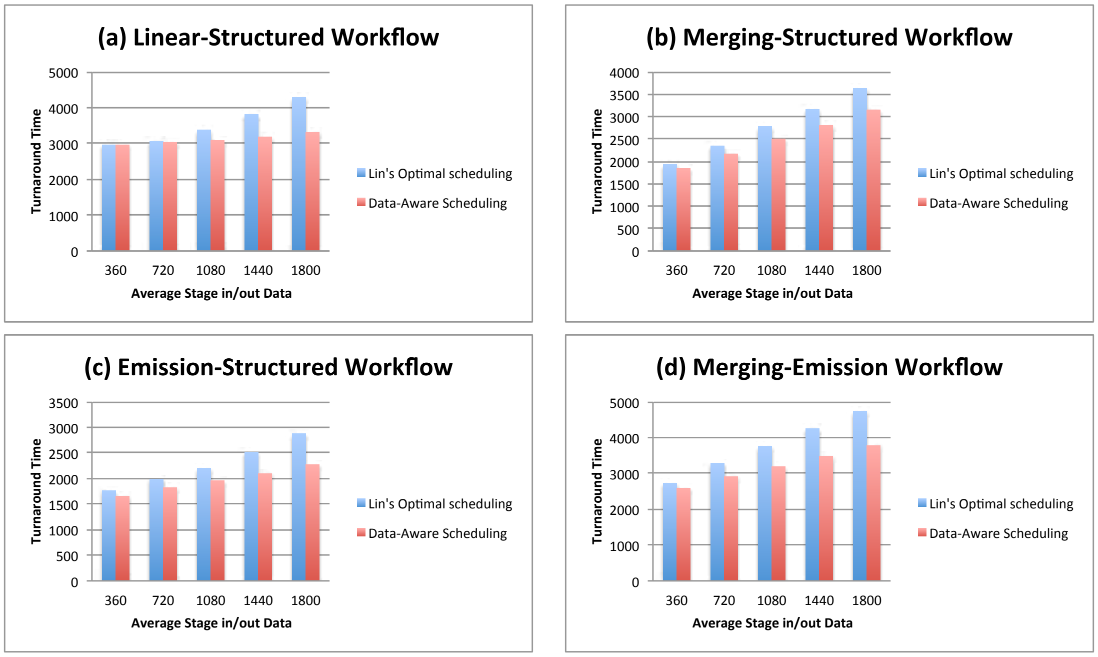


FIGURE 4.13: Turnaround time comparison for typical workflow structures

Chapter 5
Data-aware Approximate Optimal Scheduling

Workflow scheduling is a very important problem in the distributed computing and Grid computing research area. It has beed studied several decades by scientists aiming at finding an optimal solution in an efficient manner or approximating optimal solution with high accuracy. In this study, a genetic algorithm approach is proposed. Distinct from the past research, we develop a novel data-aware evaluation function for each chromosome, a common augmenting crossover operator and a simple but effective mutation operator.

5.1 Task Height

To facilitate the crossover and mutation operation, we define the task height similar to the definition in [21, 44, 52].

$$Height(t_i) = \begin{cases} 0 & if\ t_i\ is\ root \\ \max\limits_{t_j \in pred(t_i)} Height(t_j) + 1 & otherwise \end{cases} \tag{5.1}$$

From the definition, It is easy to see that if t_j is an ancestor of t_i, then $Height(t_j) < Height(t_i)$. It is always feasible that a task with a smaller height is executed before a task with a larger height and tasks with the same height are executed in an arbitrary order. However, there are some occasions that it is also feasible to execute a task with a larger height before a task with a smaller height. For example, in Figure 5.1, the height of each task is shown in Table 5.1. The height of t_2 is 1, the height of t_6, t_7, t_8 and t_9 are greater than one, yet it is still feasible to execute t_1 after them.

TABLE 5.1: Task height

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
Height	0	1	1	1	3	2	3	2	4	5
$Height_{eq}$	0	3	2	1	4	3	3	2	4	5
$Height_{soft}$	0	2	1	1	3	3	3	2	4	5

In order to have a uniform relationship between task execution order and task height, we introduce two additional height notations, $height_{eq}$ and $height_{soft}$. The height of a task is calculated topdown and the equivalent height is calculated bottom up. A soft height is defined as a value in between the

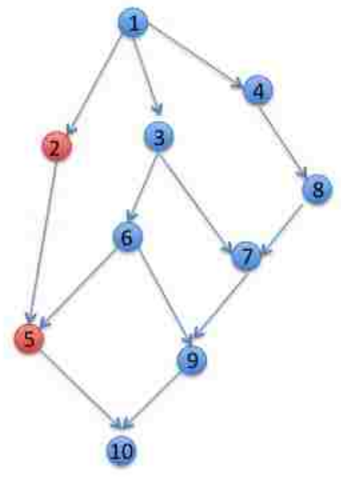


FIGURE 5.1: The workflow

original height and equivalent height. $Height_{soft}$ is the same as $Height'$ defined in [44]. The height and equivalent height of a task are fixed, while the soft height can be different for each chromosome. Since each chromosome will have a distinct task height array and the task execution orders are based on the soft height, when we have a large number of chromosomes, we can explore a broad search space for task execution orders and increase the probability of finding an optimal scheduling.

$$Height_{eq}(t_i) = \begin{cases} Height(t_i) & if\ t_i\ is\ end\ task \\ \min\limits_{t_j \in Suc(t_i)} Height_{eq}(t_j) - 1 & otherwise \end{cases} \tag{5.2}$$

$$Height_{soft}(t_i) = Height(t_i) + rand()\%(Height_{eq}(t_i) - Height(t_i) + 1) \tag{5.3}$$

5.2 Chromosome Encoding

A chromosome should contain the information of solution which it represents. For the workflow scheduling problem, the solution is a complete scheduling, a mapping from each task to a processor. For a complete scheduling, each task in the workflow will be assigned to one processor in a certain order. The tasks on each process will be execute sequentially and the tasks on different processors

TABLE 5.2: A complete scheduling

processor	task
P_1	2 5
P_2	3 6 9
P_3	4 8 7

will be executed in parallel. To represent the information of a particular scheduling, the processor on which the tasks will be executed and the execution order on each processor should be considered.

In this study, we have the following assumptions for processor IDs and task IDs.

1. The start task ID is 1 and it is a dummy task with zero execution time and zero data transfer to its successors. This will ensure the entire workflow will have a single start point.

2. The end task is a dummy task with zero execution time and zero data transfer between its predecessors. This will ensure the entire workflow will have a single end point.

3. The processor ID is an integer and greater than zero. The will enable us to user zero as a delimiter during encoding.

Since both processor ID and task ID are integer, it is very convenient for us to use integer representation during encoding. We append the ID of each task assigned to a processor to the processor ID and use zero as a delimiter for the task assignment for each processor. For example, the encoding for the scheduling in Table 5.2 is: 0 1 2 5 0 2 3 6 9 0 3 4 8 7.

5.3 Population for Workflow Scheduling

The population is a set of chromosomes. To generate a chromosome, first calculate the soft height for each task, and then assign tasks in order of the soft height randomly to a processor. Each chromosome will maintains its own task soft height array and will govern the mutation process for each chromosome. The process of generating the population is shown in Algorithm 3.

5.4 Crossover Operator

The crossover operation is performed on two chromosomes with the hope of generating new offsprings with higher fitness. The genes in the conventional chromosome are independent to each other, which makes the crossover operation simple and feasible. Taking two feasible chromosomes and intermingling

Algorithm 3 Generate population

```
1:  for $i = 1$ to $population\ size$ do
2:      calculate the soft height of each task
3:      maxHeight $\leftarrow$ the maximal soft height of the tasks for the current chromosome
4:      for $j = 1$ to $maxHeight$ do
5:          for all $task$ such that $Height_{soft}(task) = j$ do
6:              randomly assign $task$ to a $processor$
7:          end for
8:      end for
9:      encoding the complete scheduling to a $chromosome$
10:     add the $chromosome$ to the $population$
11: end for
```

their genes will still generate two feasible chromosomes. However, in the workflow scheduling problem, it is more complex. The chromosome is comprised of a set of genes representing task assignment, processor ID and delimiter. Each chromosome represents a complete scheduling and should cover all the task assignments. Simply exchanging gene segments is prone to result in incomplete chromosome and gene duplication. For example, suppose we have the following two chromosomes:

chrom1 = 0 1 2 5 0 2 3 6 9 0 3 4 8 7

chrom2 = 0 1 2 9 0 2 3 6 7 0 3 4 8 5

After exchanging the first 4 integers of these two chromosomes, we get the following two offsprings:

offspring1 = 0 1 2 9 0 2 3 6 9 0 3 4 8 7

offspring2 = 0 1 2 5 0 2 3 6 7 0 3 4 8 5

It is easy to observe that $offspring1$ is incomplete since it lacks the information of where 5 is assigned. Meanwhile, it has duplicate and inconsistent information since it indicates that 9 is assigned to both processor 1 and 2. $offspring2$ has the same problem as $offspring1$.

Past research has been done to perform crossover to tasks assigned to each processor, do some adjustment on the fly and finally output two feasible offsprings. This approach is little complex for implementation. Besides, there is no clear evidence showing that such a complex crossover outperforms a simple operation by replacing some chromosomes of small fitness values with some new fresh chromosomes.

In this work, a new approach is introduced to simplify the crossover operation. In this approach, instead of replacing two parents with two new offsprings, the population will take one new offspring generated from two parents as well as keeping the parents. In the end, the population size will increase to one and a half times of the original size. A ranking selection method can reduce the population back to its original size.

In this new approach, if a task is assigned to the same processor in the parent chromosome, it will remain this assignment in the offspring. The tasks having different assignment in the parent chromosome will have a random assignment in the offspring. For example, an offspring could be $offspring = 0\ 1\ 2\ 9\ 0\ 2\ 3\ 6\ 5\ 0\ 3\ 4\ 8\ 7$ if the parents are $chromosome1$ and $chromosome2$. The detail of the operation is shown in Algorithm 4, 5

Algorithm 4 Crossover

```
1: for all ($chrom_i, chrom_j$) such that ($chrom_i \in population$) $\wedge$ ($chrom_j \in population$) $\wedge$ ($chrom_i \notin p$) $\wedge$ ($chrom_j \notin p$) do
2:    $offspring \leftarrow GenerateOffspring(chrom_i, chrom_j)$
3:    add $offspring, chrom_i, chrom_j$ to $p$
4: end for
5: $population \leftarrow$ select the best $populationsize$ chromosomes from $p$.
```

Algorithm 5 GenerateOffspring($chrom_i, chrom_j$)

```
1: $taskToProcessor[taskCount] \leftarrow 0$
2: for all $t$ such that $t$ is assigned to the same processor in both $chrom_i$ and $chrom_j$ do
3:    $taskToProcessor[t] \leftarrow processorID$
4: end for
5: randomly select the soft height of each task from parents
6: maxHeight $\leftarrow$ the maximal soft height of the tasks
7: for $j = 1$ to $maxHeight$ do
8:    for all $t$ such that $Height_{soft}(t) = j$ do
9:       if $taskToProcessor[t] \neq 0$ then
10:         assign $t$ to $taskToProcessor[t]$
11:      else
12:         randomly assign $t$ to a $processor$
13:      end if
14:   end for
15: end for
16: encoding the complete scheduling to a $chromosome$
```

5.5 Mutation Operator

The mutation operation in the workflow scheduling problem should ensure the chromosome after mutation to be a complete scheduling without redundancy. Besides, the task execution order should follow the data dependency constraints. Simply changing a task ID will cause the chromosome to contain duplicate and inconsistent information. Moreover, the chromosome after this mutation is incomplete since it lacks the task assignment for the task before mutation. Randomly exchange the position of two tasks might violate the data dependency constraints.

In this work, a simple and effective mutation method is introduced. First randomly choose two processors and perform a merge sort to the tasks according to their soft height, then randomly assigned each task to these two processors sequentially. Algorithm 6 shows the detail of the mutation operation.

Algorithm 6 Mutation

```
1: randomly choose two processors $i$ and $j$.
2: perform a merge sort to the tasks assigned to processor $i$ and $j$ according to their soft height
3: for all task in the sorted task array do
4:     randomly assign this task to processor $i$ or $j$
5: end for
```

5.6 Fitness Function for Scheduling

The fitness is calculated based on the turnaround time of the scheduling represented by the chromosome. Since the objective is to find a scheduling with the shortest turnaround time, a chromosome with shorter turnaround time will have higher fitness value. The fitness in this work is calculated by the difference of the maximal turnaround time in the population and the turnaround time of the current chromosome.

To get the turnaround time of scheduling represented by the chromosome, the task assignment and execution order must be retrieved from the chromosome. The tasks assignment on each processor is already sorted according to the soft height. Performing a merge sort to the task assignments on each processor will output an array of task assignments in nondecreasing order of soft height.

5.7 Experimental Results

Algorithm 7 RetrieveSched

```
$sched \leftarrow$ task assignment on processor 1
**for** $pid = 2$ to $processor\ count$ **do**
    merge tasks assigned to processor $pid$ with $sched$ according to softheight
**end for**
**for all** task in $sched$ **do**
    calculate the earliest stage in time
    calculate the earliest start time
    calculate the earliest time to transfer data
    calculate the earliest stage out time
**end for**
**for all** processor **do**
    calculate the finish time of the last task assigned to this processor
    calculate the stage out finish time time of each task which has data to be staged out
**end for**
return the maximum of latest execution finish time and latest stage out finish time
```

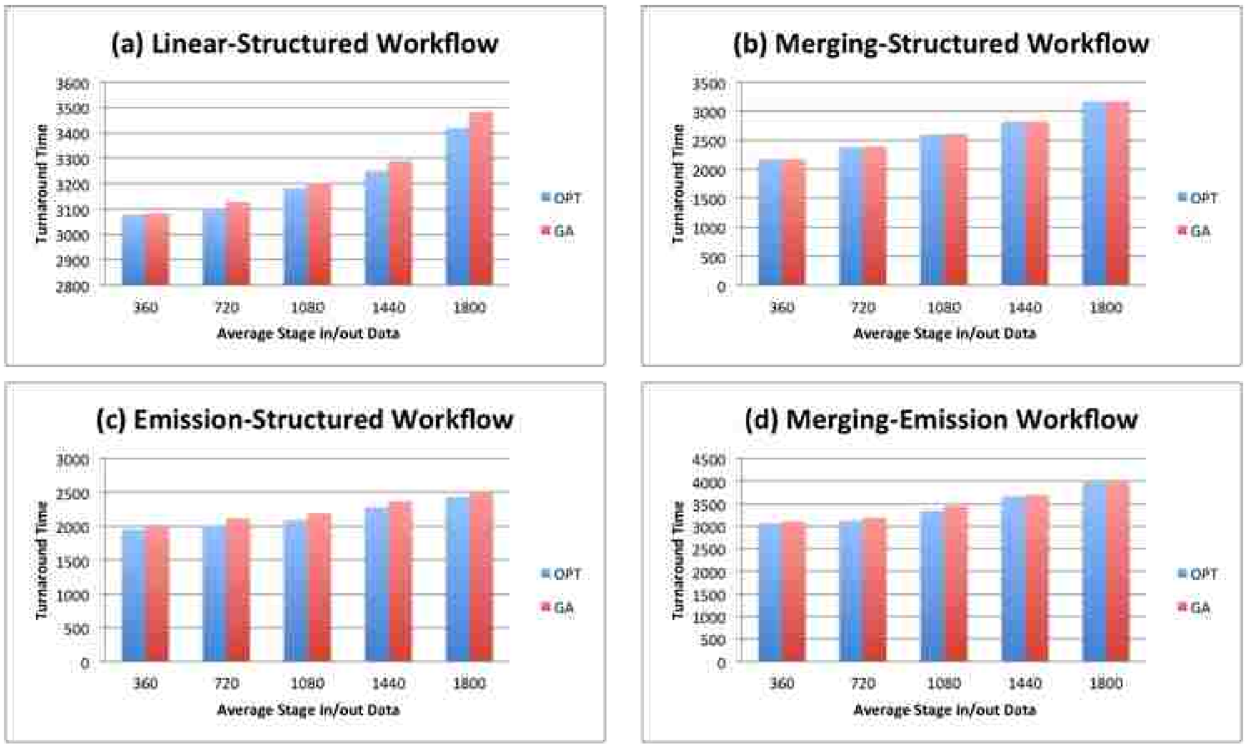


FIGURE 5.2: Turnaround time comparison of GA and optimal scheduling

Chapter 6

Parameterized Model for Optimizing Workflow Data Throughput

6.1 Review of Existing Prediction Models

6.1.1 Dinda et al Model

In this model, the throughput(Th_n) for n parallel streams is predicted in the following equation by Dinda et al [32]. Note that there are two parameters on the right side of the equation, which are separately a' and b'.

$$Th_n = \frac{n}{\sqrt{p'_n}} = \frac{n}{\sqrt{a'n^2 + b'}} \tag{6.1}$$

6.1.2 Logarithmic Model

This model is proposed by Yildirim et. al [49]. There are also two parameters in this model, which are separately a' and b'. The only difference is the term in the denominator compared with the Dinda et al Model. In this model, the square root term in the denominator is replaced with an exponential term.

$$Th_n = \frac{n}{\sqrt{a'e^{b'n}}} \tag{6.2}$$

6.1.3 Newton's Model

This model is also proposed by Yildirim et al [49]. There are three parameters in this model, which are separately a', b' and c'. It differentiates from Dinda et al model by placing an extra parameter c' in the denominator. Recall that in Dinda et al model, they assume that the highest order of the tern in the square root is n^2. In the Newton's model proposed by Yildirim et al, the highest order is relaxed by placing a new parameter c'. They intend to improve the precision of the prediction model by finding the best order with respect to n.

$$Th_n = \frac{n}{\sqrt{a'n^{c'} + b'}} \tag{6.3}$$

6.2 An Improved Model: Full Second Order Model

By examining Dinda et al's Model, we find that the square root terms in the denominator are partial second order. An intuitive idea is adding a linear term b' with respect to n in the square root. Then there will be there parameters, which are separately a', b' and c'. We have done some experiments on the Newton's method mentioned above and find that the best value for c' in equation 6.3 is very close to 2 (2 ± 0.2). Hence in this proposed new model, we fix the highest order with respect to n in the square root as 2. The theoretical study has been published in [49] and it has been widely used in our fast data placement models [13] [12].

$$Th_n = \frac{n}{\sqrt{p'_n}} = \frac{n}{\sqrt{a'n^2 + b'n + c'}} \tag{6.4}$$

Similar to the idea presented in [49], in order to instantiate the throughput prediction model, we need to calculate the parameters. Since there are three parameters in this full second order model, if we treat the number of parallel streams n and the the throughput th_n corresponding to n as known data, we can build an system of equations from a set of data pairs $(n_1, thn_1), (n_2, thn_2), (n_3, thn_3)$ and derive the parameters as the following.

$$a' = \frac{\frac{\frac{n_3^2}{Th_{n_3}^2} - \frac{n_1^2}{Th_{n_1}^2}}{n_3 - n_1} - \frac{\frac{n_2^2}{Th_{n_2}^2} - \frac{n_1^2}{Th_{n_1}^2}}{n_2 - n_1}}{n_3 - n_2} \tag{6.5}$$

$$b' = \frac{\frac{n_2^2}{Th_{n_2}^2} - \frac{n_1^2}{Th_{n_1}^2}}{n_2 - n_1} - (n_1 + n_2)a' \tag{6.6}$$

$$c' = \frac{n_1^2}{Th_{n_1}^2} - n_1^2 a' - n_1 b' \tag{6.7}$$

6.3 Randomly Chosen Data Sets for Comparison

To evaluate the performance of different models by using some specified data set may not give us a good approximation of the power of the models. In existence of the appropriate combination of data sets, a model may give less error rates. However, for different sets of data, while some models give accurate results the others may not. In the previous experiments we assumed that we have the total throughput values of all parallelism levels, in our case, up to 40 streams. However, in practical cases, it might not be possible to gather the whole data set information. The problem is whether we can use the limited number of data to find the suitable coefficients for the prediction models. In this section, we design an evaluation strategy which will use randomized datasets that will cover subsets of all

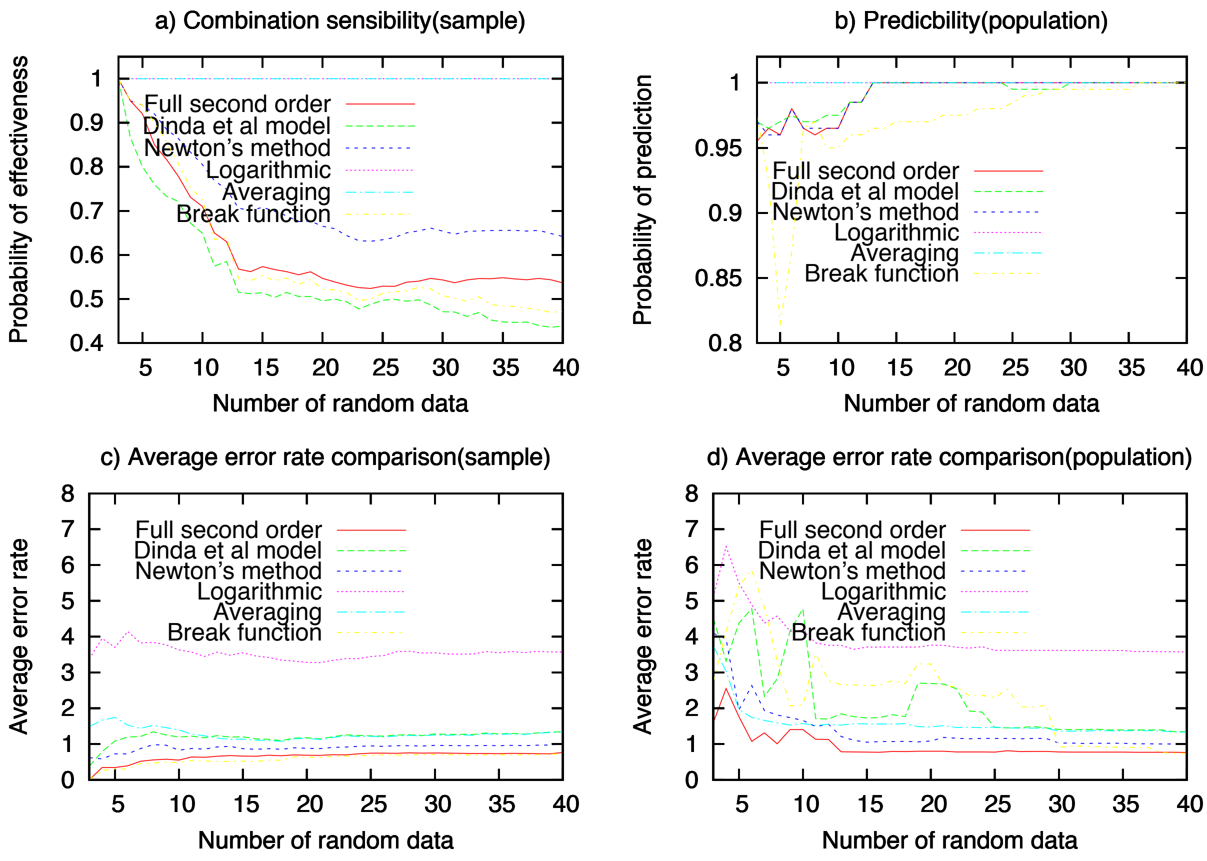


FIGURE 6.1: Model evaluation based on predictability, sensibility and error rate.

the parallelism level information. Based on this strategy we also define new metrics and evaluate our models based on them.

Since the number of data used for model evaluation will affect the accuracy of our prediction, we will try different numbers of data in the evaluation procedure. We use a data file that keeps all the data we get from the experiments we have conducted. We call all the data in this file which includes the throughput values of all parallelism levels up to 40 the *population*, from which we get *sample*s with different numbers of data for our evaluation procedure. The sample data sets are taken from the population randomly.

6.3.1 Comparison Metrics

For each sample data set, we calculate the coefficients and get the equation to predict the throughput. The coefficients are obtained from the sample data set, however, we will make the comparison based on both the data from the *population* and the data from the *sample*.

The predicted throughput can not be calculated for some paralelism levels with the coefficients derived from a specified stream combination selection as they will make the term of the square root less than zero or make the denominator equal to zero. So we call those stream numbers out of the domain. We say that the *Predictability* of a specified equation is poor if the number of streams out of the domain is large in proportion. We use $P_{pre[m\_i]}$ to stand for the predictability of a specified equation derived from the data set named m_i where m represents the number of data and i represents the ith set. Let the number of streams within domain be represented by N_{ind} and the number of streams out of domain be represented by N_{outd}, then we define $P_{pre[m\_i]}$ as follows:

$$P_{pre[m\_i]} = \frac{N_{ind}}{N_{outd} + N_{ind}} \tag{6.8}$$

The equation solved by some combinations of parallelism level data will cause the predictability to be in a poor situation, while some other combinations may cause the predictability to be stronger. If there are lots of combinations of data which make the predictability to be in a poor situation, then we say that this prediction model is sensitive to data. Let $P_{sen[m\_i]}$ to be denoted as the sensitivity of the sample data set, the number of combinations which make $P_{pre[m\_i]} = 1$ to be denoted as N_{eff} and the number of all the combinations of a sample data to be denoted as N_{sam}, then we calculate $P_{pre[m\_i]}$ as follows:

$$P_{sen[m\_i]} = \frac{N_{eff}}{N_{sam}} \tag{6.9}$$

For evaluation of accuracy of the models, we use the quadratic average of the distance metric that was used in BFC Algorithm for stream values within the domain.

$$Err_{[m\_i]} = \sqrt{\frac{\sum_{k=1}^{N_{ind}} \epsilon_k^2}{N_{ind}}} \tag{6.10}$$

We propose to find a best fitted combination of data for the data set m_i such that $P_{pre[m\_i]} = 1$ and $Err_{[m\_i]}$ gets the minimum value compared with all possible combinations for the sample data. The following equation represents the best error rate value among different combinations of parallelism levels.

$$Err_{best[m\_i]} = min_{N_{eff}} \left(Err_{[m\_i]}\right) \tag{6.11}$$

Finally, by taking the average of results for all random data sets, we compare our models. In the following equations k_m represents the number of data sets of m data.

$$P_{sen[m]ave} = \frac{\sum_{i=0}^{k_m-1} P_{sen[m\_i]}}{k_m} \tag{6.12}$$

$$P_{pre[m]ave} = \frac{\sum_{i=0}^{k_m-1} P_{pre[m\_i]}}{k_m} \tag{6.13}$$

$$Err_{best[m]ave} = \frac{\sum_{i=0}^{k_m-1} Err_{best[m\_i]}}{k_m} \tag{6.14}$$

6.3.2 Evaluation

In this section, we present the results based on the evaluation metrics we defined above. Figure 6.1.a shows the sensibility of data that is used for prediction using all models as the number of data is increased in the samples. We select one combination of parallelism level data each time to get the coefficients of the throughput equation. The y-axis is the probability of the effectiveness of these combinations. If the equation derived from a specified combination can predict the throughput of each number of stream, i.e., $P_{pre[m\_i]} = 1$, we say this combination is effective in the data set m_i. The sensibility is inversely related to the effectiveness of the combinations. From Figure 6.1.a we can see that the logarithmic and the averaging model is not sensible to the data. So they are the most stable models. Newton's Iteration follows them with a high probability of effectiveness. Full second order, break function and Dinda et al. models are the most sensible to data. Another observation is that as the number of data increases, the combinations become more sensitive to data.

Figure 6.1.b presents the predictability of a throughput equation with specified coefficients. Once the equation is applied with the coefficients, the throughput value for some number of streams can not be calculated as the denominator becomes equal to zero or the term inside the square root is less than zero. In this case, we say that those stream numbers are out of the domain. We can not predict the throughput of a stream number if it is out of the domain. From the figure, it can be seen that logarithmic and averaging models are the most predictable while Newton's Iteration Model, Dinda et al Model and Full Second Order Model can only reach them for large number of data. The worst performance is with the break function model.

Figure 6.1.c presents the average error for the population data, in which case the predicted throughput is calculated for every parallelism level and compared to the whole data set of experimental results. According to the figure, the best results are taken with Full Second Order and Newton's Iteration and the worst ones are taken with logarithmic model.

When the predicted results are calculated for only the parallelism level existing in the sample, break function model gives the best results and Full Second Order and Newton's Iteration follows it (Figure 6.1.d). The worst are again taken with logarithmic model. The results are similar when comparison is made for population and sample data sets. This is a strong proof of the correctness of our scheme of using a subset of the pupulation data to predict the throughput of the population.

6.4 Decreasing Dataset Size with Intelligent Selection of Parallelism Levels

In the previous evaluation sections, we have presented the power of the models for randomly chosen parallelism levels and randomly chosen data sets. When we choose a random combination of parallelism levels, it is a high possibility that the chosen levels may not reflect the characteristics of the throughput curve. The algorithm we have presented solves this problem by finding the best coefficients. However it tries a large number of combinations to reach a steady result and it may need a certain size of historical data set to give a good prediction hence can not be used with an online strategy which uses the prediction results of tools rather than past historical transfers.

In this section, we provide an intelligent selection strategy which decides on less number of data and can be used with an online model as well. Our previous experiences showed that it is better if we choose the parallelism levels not close to each other. So we applied an exponential increase strategy by selecting the stream numbers that are power of 2: $1, 2, 2^2, 2^3, ..., 2^k$. Each time we double the number of streams until the throughput starts to drop down or increase very slowly compared to the previous level. After k+1 steps we gather k+1 parallelism level throughput data and apply the Best-fitted Coefficient Algorithm to find the best parallelism levels. In this case we only need a data size of O(log 2^k) and hence do less number of comparisons.

This strategy (best fit coefficient) focuses on better prediction of the optimal parallelism level which is the peak point of the curves. While the basic algorithm gives better results for the average throughput distances in most of the cases, with exponential increase strategy we are able to better predict the peak throughput point hence the number of streams that gives it. More importantly, we could find this point with a small size dataset of exponentially increasing points. According to the figure, only 4 points (1,2,4 and 8) seems to be enough to get a good prediction curve. Best results are taken with Full Second Order model in terms of both throughput distance and peak point prediction. Newton's Iteration and Logarithmic models can predict the peak point well, however with a trade of high throughput distance between GridFTP and prediction results.

6.5 Validating Prediction Model

To figure out whether a certain combination of data points is good or not before we try to use it to calculate the optimum parallel stream numbers, we examine the coefficients a',b' and c' derived from application of the model by using the certain combination of data points. If the coefficients meet some requirements for each model, then we can use this combination to calculate the optimum parallel stream number, otherwise we need to get another combination of data points. But we have to keep in mind that, we may still be far from the combination that will give the most accurate model.

In the following subsections we give out the requirements of the coefficients that should be met so that they can correctly reflect the relationship between the throughput and parallel stream number for the models. Since we have based our model improvements on Dinda et al model [32], we consider this model as a base while comparing our models. Also we present a short proof for the Full Second Order Model.

6.5.1 Statements of the Coefficients Requirements

i)For Dinda et al model:

- $a' > 0$
- $b' > 0$

ii)For Newton's Method Model:

- $a' > 0$
- $b' > 0$
- $c' \geq 2$
- $\left(\frac{2b'}{a'(c'-2)}\right)^{\frac{1}{c'}} > 1$

$iii)$For Full second order Model:

- $a' > 0$
- $b' < 0$
- $c' > 0$
- $2c' + b' > 0$

6.5.2 Proof of Dinda et. al Model Parameter Requirements

$$\begin{aligned}
Th'_{din} &= \frac{b'}{(a'n^2+b')^{\frac{3}{2}}} > 0 \\
&\Rightarrow \begin{cases} b' > 0 \\ a'n^2 + b' > 0 \quad \forall n \in N^+, n \leq optnum \end{cases} \\
&\Rightarrow \begin{cases} a' > 0 \\ b' > 0 \end{cases}
\end{aligned}$$

Furthermore,

$$\begin{aligned}
\lim_{n\to\infty} \frac{n}{\sqrt{a'n^2+b'}} &= \lim_{n\to\infty} \frac{\sqrt{a'n^2+b'}}{a'n} \\
&= \lim_{n\to\infty} \frac{\sqrt{a' + \frac{b'}{n^2}}}{a'} \\
&= \frac{\sqrt{a'}}{a'} \qquad (6.15)
\end{aligned}$$

Then we conclude that the throughput function of *Dinda et. al* Model increases monotonically with an upper bound $\frac{\sqrt{a'}}{a'}$.

6.5.3 Proof of Newton's Method Model Coefficients Requirements

For Newton's Method Prediction Model, we expect a prediction curve that will increase first to a peak value, and then decrease gradually to a lower bound.

Based on Equation 6.3, we know that $c' \geq 2$, otherwise, the limit of the throughput function will be infinity. When $c' = 2$, the throughput equation will be the same as the Dinda et al Model. From the discussion about Dinda et al Model, we know that the throughput function of that model increases monotonically. In this case, in order to get a throughput prediction curve with a peak value c' must be greater than 2.

Also we must ensure that $a' > 0$, otherwise, the term inside the square root will be less than zero.

To meet the increasing and decreasing properties of the throughput curve, the first derivative should be positive initially, in the peek point it should become zero, and finally become negative. Since the denominator is positive, we only need to control the numerator of the throughput function. If $b' \leq 0$, the numerator will be negative, thus we conclude that $b' > 0$.

When we equalize the numerator to 0, we find that $n = \left(\frac{2b'}{a'(c'-2)}\right)^{\frac{1}{c'}}$. Since the throughput will increase along with the stream number firstly. So the optimum value of n should be greater than 1. Thus we get $\left(\frac{2b'}{a'(c'-2)}\right)^{\frac{1}{c'}} > 1$.

6.5.4 Proof of Full Second Order Model Coefficients Requirements

In order to get a curve which increases first and then decreases monotonically, we should guarantee that the throughput function is positive first and when it reaches the peak point, becomes zero, finally decreases into negative value.

$$Th'_{ful} = \begin{cases} \frac{\frac{b'}{2}n+c'}{(a'n^2+b'n+c')^{\frac{3}{2}}} > 0 & n < optnum \\ \frac{\frac{b'}{2}n+c'}{(a'n^2+b'n+c')^{\frac{3}{2}}} = 0 & n = optnum \\ \frac{\frac{b'}{2}n+c'}{(a'n^2+b'n+c')^{\frac{3}{2}}} < 0 & n > optnum \end{cases}$$

$$\Rightarrow \begin{cases} \frac{b'}{2}n + c' > 0 & n < optnum \\ \frac{b'}{2}n + c' = 0 & n = optnum \\ \frac{b'}{2}n + c' < 0 & n > optnum \\ a'n^2 + b'n + c' > 0 & \forall n \in N^+ \end{cases}$$

$$\Rightarrow \begin{cases} a' > 0 \\ b' < 0 \\ c' > 0 \\ optnum = \frac{-2c'}{b'} > 1 \end{cases}$$

$$\Rightarrow \begin{cases} a' > 0 \\ b' < 0 \\ c' > 0 \\ 2c' + b' > 1 \end{cases}$$

Furthermore

$$\begin{aligned} \lim_{n\to\infty} \frac{n}{\sqrt{a'n^2+b'n+c'}} &= \lim_{n\to\infty} \frac{2\sqrt{a'n^2+b'n+c'}}{2a'n+b'} \\ &= \lim_{n\to\infty} \frac{\sqrt{a'+\frac{b'}{n}+\frac{c'}{n^2}}}{a'} \\ &= \frac{\sqrt{a'}}{a'} \end{aligned} \tag{6.16}$$

According to the above equalities and inequalities we conclude that *Full second order* Model increases first, then reaches a peak value, later decreases with a lower bound $\frac{\sqrt{a'}}{a'}$.

Chapter 7
Regression Model for Optimizing Workflow Data Throughput

Along with the advance of high throughput computing (HTC), high performance computing (HPC), and Many-task computing (MTC), more attention has been drawn to reduce the impact on the overall performance caused by the low data transferring rate. In the grid environment, data can be transferred with parallel TCP streams in order to fully utilize the underlying network bandwidth. It is crucial to figure out the optimal number of parallel streams efficiently and effectively. In this study, dynamic models and stochastic models based on data mining models such as linear regression and Gaussian process regression are introduced and compared. Also a framework to predict the optimal number of parallel streams is designed and implemented, which is feasible to variate models presented in this study.

7.1 Introduction

Gridftp is widely used in the grid environment for data movement due to its high transferring rate and the reliability. Globus is a toolkit using gridftp as its underlying protocol for data movement. In the implementation of Globus, it allows the user to specify the number of parallel TCP streams in order to fully utilize the network bandwidth. There exists an optimal number with which the throughput reaches the peak value. Transferring with a number less than the optimal value will result in a drastically lower throughput. On the other hand, transferring with a number greater than the optimal value will not lead to higher throughput; nevertheless, it will incur a higher resource utilization of the system. The Globus toolkit does not provide the user with the optimal value automatically since it fluctuates along with the changing of the network traffic. Hence it is crucial to predict the optimal value based on some effective mathematical models.

In the study [32], a partial second order model was introduced to model the throughput variance with respect to increasing the number of parallel streams. In this model, the throughput increases drastically when the number of parallel streams is small, and then it becomes stable. The throughput is monotonically increasing with respect to the number of parallel streams according to the model presented in [32]. However, this model is biased since in most cases the throughput does not exhibit monotonically increasing behavior. Alternatively, the throughput tends to decrease gradually when the number of parallel streams exceeds a certain number, which is the optimal number of parallel streams.

A full second order model was introduced to improve the accuracy of the partial second order model in the previous chapter. In the full second order model, p'_n [32] is related to a full second order polynomial, other than the partial second order one which was presented before. The full second order model can predict the first increasing and then decreasing behavior of GridFTP throughput as the number of streams increases. Experiments show that full second order can give better results than the partial order.

7.2 Problems of the Three-point Prediction Approach

7.2.1 Sensibility of Data Selection

As presented above, the full second order model requires three data pairs to calculate the unknown variables in order to instantiate the model. Unfortunately, the choice of these three data pairs affects the prediction results significantly. Some combination of three data pairs is able to construct a very accurate model, while others not. Even worse, it is possible that a improperly chosen combination of data pairs does not give a correct model.

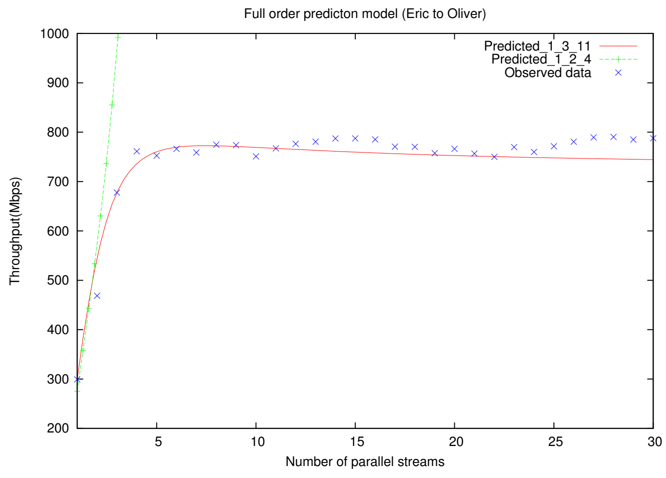


FIGURE 7.1: Sensibility of data pairs

Figure 7.1 depicts the sensibility of data pairs selection. The horizontal axis is the number of parallel streams and the vertical axis is the corresponding throughput. The crossed points shown in the figure represents the observation data, which are measured between two clusters (Eric and Oliver) residing in the LONI optical network. The curve produced by using the throughput with respect to the parallel number of streams of 1, 3 and 11 agrees with the observation data very well. However, the curve produced by using the throughput corresponding to the number of parallel streams of 1, 2 and 4 deviates from the observations drastically.

7.2.2 Time Efficiency of Sampling

In order to overcome the sensibility problem, more data are measured in the sampling process instead of just providing three data pairs. An effective way is to measure the throughput with respect to an exponentially increasing number of parallel streams until the throughput does not increase compared with the previous sampling. For instance, gradually measure the throughput with respect to 1, 2, 4 and 8. If the throughput with respect to 8 is less than the throughput of 4, then stop sampling. Choose the best combination from these 4 observations to construct a model which can be used to predict the optimal number of parallel streams. This strategy could decrease the prediction failure rate. However, it will increase the time spent on sampling.

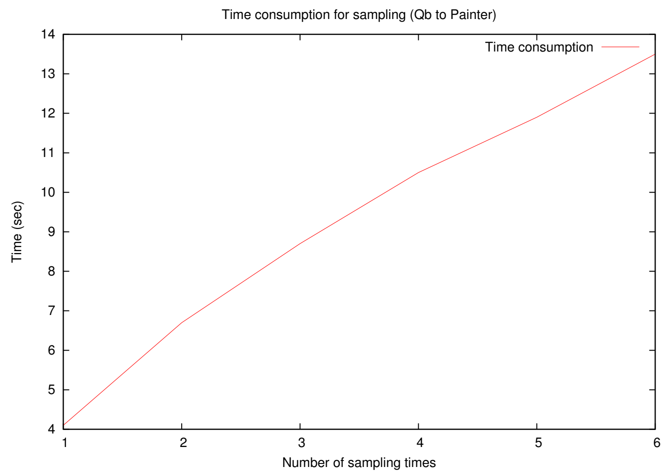


FIGURE 7.2: Time consumption for sampling between LONI clusters

Figure 7.2 shows the time consumption with respect to the number of sampling times. The file size for each sampling is 30MB and the number of parallel streams is exponentially increased. It costs around 7 seconds when the number of sampling times is 2 and 14 seconds when the number of sampling times is 6. The reason that the time consumption is not linear with respect to the number of sampling times is due to the exponentially increasing of the parallel streams. Generally the time cost will be less with a greater number of parallel streams for the same size of data to transfer.

7.2.3 Fluctuation of the Network

The fluctuation of the network increases the difficulty of prediction. The basic idea of the prediction model presented above is predicting the optimal number of parallel streams using the throughput with respect to different number of parallel streams at a fixed time t. However, actually only one throughput with respect to a certain number of parallel streams can be measured at time t. The other sampling data are measured at different time. Hence the existing prediction models assume that the throughput does not change along with the time even though it does.

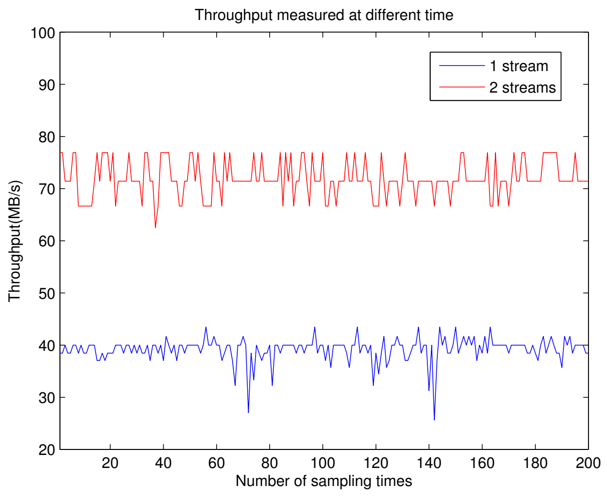


FIGURE 7.3: Throughput measured at different time between LONI clusters

Figure 7.3 shows the fluctuation of the network measured at different time with respect to 1 and 2 parallel streams respectively. They are measured for 200 times with a 20 seconds time interval. The throughput changes each interval for both 1 and 2 parallel streams.

Figure 7.4 shows the average throughput with respect to different number of parallel streams and one specific case. The average throughput curve is smooth and easier to construct a prediction model based on these observations. However, it is harder to construct a effective model based on the data

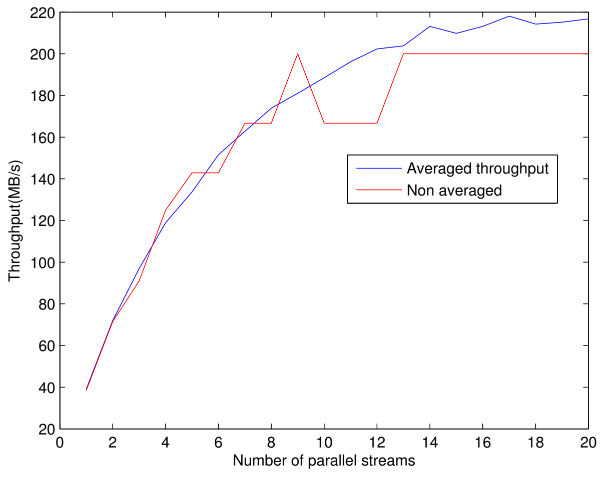


FIGURE 7.4: Average throughput vs Non-average throughput between LONI clusters

of the specific case. Nevertheless, we can only obtain the data for one specific case when we try to construct the prediction model. That increases the data sensibility during model construction.

7.3 Data Mining Approaches

7.3.1 Linear Regression

The objective of the prediction is to find the optimal number of parallel streams that makes the throughput maximal at a given time. In order to achieve this goal, a naive approach could be: first measure the throughput for each number of parallel streams and then figure out the one whose throughput is the largest. The number corresponding to the largest throughput is claimed to be the optimal value. Apparently this naive approach guarantees the accuracy, however, it is very expensive in terms of time cost as presented in the previous sections.

Linear regression is widely used in solving data mining problems where the output depends linearly on the input. In order to use linear regression approach, it is assume that the optimal number of parallel streams depends linearly on the throughput of several number of parallel streams.

7.3.1.1 Basic Idea

The basic idea of this approach is using the history information to make a prediction of a new observation based on some criteria. There exists some patterns concealed in the history information that are not obviously. In order to make a accurate prediction about the new observation, one needs to select the most significant features from the history data. Specifically, in the optimal number of parallel streams prediction problem, it is assumed that the optimal number is linearly related to a set of throughput with respect to a set of number of parallel streams. Suppose there are k elements in this set and use X_i to represent the throughput with respect to the set. opt_i represents the optimal number of parallel streams with respect to the set. W is a column vector which consists of the weight for each throughput.

$$X_i = [th_{i1}, th_{i2}, \cdots, th_{ik}]^T \tag{7.1}$$

$$W = [w_1, w_2, \cdots, w_k]^T \tag{7.2}$$

$$opt_i = \sum_{j=1}^{k} w_j * th_{ij} = W^T X_i \tag{7.3}$$

In this model, the throughput of a subset of the number of parallel streams is measured instead of measuring the throughput for each number. Each data point is a multidimensional variable which consists of the throughput with respect to a set of number of parallel stream. Suppose there are n such data from the history data. X represents all the data points from the history data, which is a k by n matrix. opt represents the corresponding optimal number of parallel streams, which is a column vector.

$$X = [X_1 \ X_2 \ , \cdots , \ X_n] \tag{7.4}$$

$$opt = [opt_1 \ opt_2 \ , \cdots , \ opt_n]^T \tag{7.5}$$

If there is a column vector W such that:

$$opt = W^T X \tag{7.6}$$

then this W is the optimal solution to this prediction model. In this situation, W perfectly agrees with each data point. However, this rarely happens in the reality since the optimal number of parallel streams is not exactly linearly related to the observed throughput. Hence a noise term σ is added into the linear model.

$$opt_i = \sum_{j=1}^{k} w_j * th_{ij} + \sigma = W^T X_i + \sigma \tag{7.7}$$

In the real world, noise often follows a Gaussian distribution. Hence it is intuitive to model opt_i as Gaussian distribution with mean $W^T X_i$ and variance σ.

$$p(opt_i | X_i, W) = N(W^T X_i, \sigma) \tag{7.8}$$

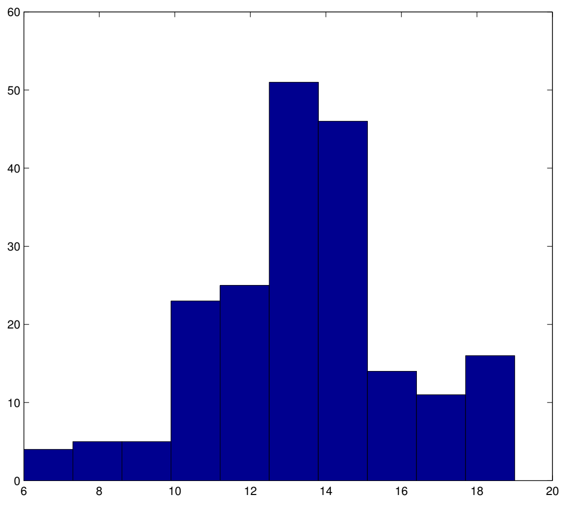


FIGURE 7.5: Distribution of the optimal number of parallel streams

This assumption also apples in this prediction model. Figure 7.5 depicts the distribution of the optimal value. In Figure 7.5, the horizontal axis represents the optimal number of parallel streams and the vertical axis represents the frequency observed for each optimal value. It turns out the probability density function appears bell shape, which approximates to Gaussian distribution.

With the assumption that the optimal number of parallel streams are independent with each other when measured at different time and they follow the identical normal distribution, the data likelihood of the n observations from the history data are calculated as follows:

$$p(opt_1, opt_2, \cdots, opt_n) = \prod_{i=1}^{n} p(opt_i) =$$
$$\prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma}} exp\left\{-\frac{1}{2\sigma}(th_i - W^T X_i)^2\right\} \tag{7.9}$$

The log likelihood is calculated as follows:

$$ln(p(opt)) = -\frac{n}{2}ln(2\pi\sigma) - \frac{1}{2\sigma}\sum_{i=1}^{n}(th_i - W^T X_i)^2 \tag{7.10}$$

The objective of linear regression model is to find a W that best suits the data X and *opt*. This means to minimize the non-consistence between the target value and the predicted value. This is equivalent to maximize the data likelihood or log likelihood. Usually a good prediction model consists of two features, one of which is accuracy and the other is simplicity. A complex model usually leads to over fitting problems. Hence a regularization term is introduced to the linear model.

$$E = E_D + \lambda E_W \tag{7.11}$$

$$E_D = \frac{1}{2}\sum_{i=1}^{n}(th_i - W^T X_i)^2 \tag{7.12}$$

$$E_W = \frac{1}{2}W^T W \tag{7.13}$$

E_D comes from the log likelihood. Notice that:

$$\frac{\partial p(ln(opt))}{\partial W} = -\frac{1}{\sigma}\frac{\partial E_D}{\partial W} \tag{7.14}$$

E_W expects the prediction model to be simple. It means it wants each w_i to to smaller. λ balances the these two terms. On one hand, it tries to make the model more accurate, and on the other hand it expects the model to be as simple as possible. To minimize E, its partial derivative with respect to W is calculated.

$$\frac{\partial E}{\partial W} = 0 \tag{7.15}$$

Solve Equation 7.15, get the optimal solution for W:

$$\hat{W} = (\lambda I + X^T X)^{-1} X^T opt \tag{7.16}$$

7.3.1.2 Mapping Function

In the linear regression model, in order to model a non linear relationship or improve the accuracy it usually maps the independent variables X into another space. The mapping function is assumed, tested and improved in each domain. In this prediction model, intuitively the optimal value is smaller if the ratio of the throughput with respect to a larger number of parallel streams to that of a smaller one is larger. For instance, at time i, the data point obtained is $X_i = (th_{i1}, th_{i2}, th_{i3})$ and at time j the data point obtained is $X_j = (th_{j1}, th_{j2}, th_{j3})$. If th_{i2}/th_{i1} is greater than th_{j2}/th_{j1}, then it is very possible that the optimal value corresponding to X_j is greater than that of X_i. Taking this into account, a function mapping from the original throughput feature space into a new ratio based feature space is introduced in Equation 7.17. Equation 7.19 is the solution to the parameter W. Equation 7.20 is the decision function of the regression model.

$$\Phi(X_i) = [X_{i2}/X_{i1} \; X_{i3}/X_{i2} \; \cdots \; X_{ik}/X_{ik-1}]^T \tag{7.17}$$

$$\Phi = [\Phi(X_1) \;\; \Phi(X_2) \;\; \cdots \;\; \Phi(X_n)] \tag{7.18}$$

$$\hat{W} = (\lambda I + \Phi^T \Phi)^{-1} \Phi^T opt \tag{7.19}$$

$$y_{prediction} = \hat{W}^T \Phi(X_{observation}) \tag{7.20}$$

7.3.1.3 Feature Selection

When moving data from one site to another, a smaller number of parallel streams is tentatively used to transfer partial of the whole data. Then increase this number by a small step and perform the second tentatively transferring and so forth. These tentatively number of parallel streams and the corresponding throughput will be used as the features for training and prediction.

Each tentative partial data movement will cost more time than using the optimal value, hence the number of tentative transfer should be as small as possible in order to minimize the time cost on the sampling phase. On the other hand, a relatively larger number of features will give a better classification model. The precision and recall of the model will be relatively better.

It is really very hard to construct a good model that using only one tentative partial data movement. However, a model with 2 tentative transfer can give a relatively good model. For instance, they can obtained by transferring the data using 1 and 4. It is interesting to figure out Which two tentative transfers can give the best model.

7.3.1.4 Prediction Comparison

In order to predict the optimal number of parallel streams, the full second order model first constructs a throughput function, and then calculate the rate of slop for each point. The point whose slop approximates to zero is recognized as the optimal solution. This idea is workable theoretically while it is difficult in practice. That is because it is very hard to decide the threshold for the rate of slop where the optimal solution relies on. A specific threshold could be suitable for one case while not for the other cases. Figure 7.6 shows the predicted results using full second order. The dots represent the predicted optimal value for each test case. The stars represent the actual optimal solution. Many predicted values are much larger than the actual ones as shown in the figure. That is because the threshold for the slop is too small for these cases, where a larger number of parallel streams is needed to make the slop reaches that threshold. A larger threshold may make these prediction values approximate to the actual value. However, the originally correct predicted value will become smaller meanwhile. Hence they will be too small than the actual value. In this situation, a fixed threshold is hard to satisfy all cases and it is hard to decide a adaptive threshold.

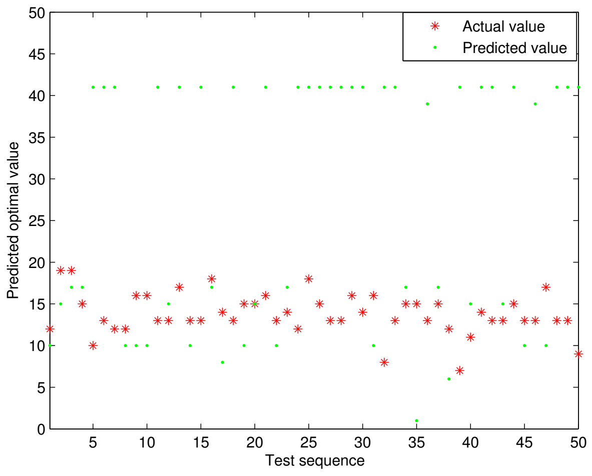


FIGURE 7.6: Predict the optimal number of parallel streams with full second order model

As contrasted to the prediction with full second order model, the linear regression model does not have such issues. The model is feasible to make a prediction for each case with a smaller error rate even using only two sampling data points. The prediction accuracy will increase when more data points are provided. Figure 7.7 illustrates the prediction using two data points which correspond to the throughput measured with the number of parallel streams of 2 and 4 respectively. Figure 7.8 represents the prediction results using 20 data points as a contrast. In Figure 7.7 the predicted value approximates to the mean of the actual value which is the most possible value for each case. In Figure 7.8 the predicted value approximates to the actual value much closer in most cases since more data points are provided. The extra information leads the predicted values closer towards the actual value. In practice a smaller number of data points is preferred other than a greater number such as 20 with the consideration of sampling efficiency. Further more, a small error in the prediction of the optimal number won't degrades the performance significantly since the throughput measured with different number of parallel streams near the optimal number approximates to each other very much.

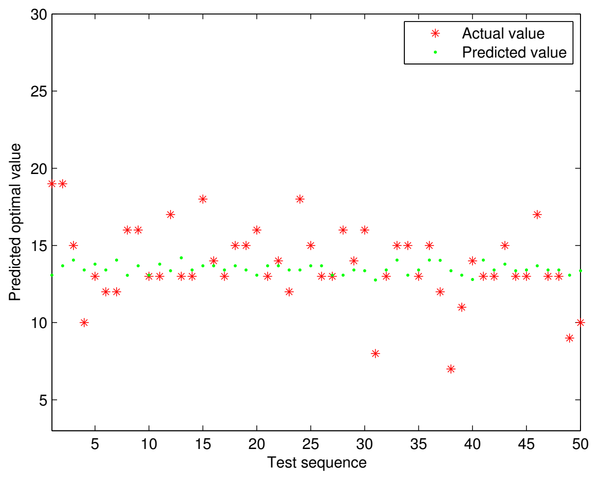


FIGURE 7.7: Predict the optimal number of parallel streams with linear regression model using two data points [2 4]

7.3.2 Gaussian Process

7.3.2.1 Function Space View

Gaussian process is defined on a collection of random variables. In this prediction domain, the optimal number of parallel streams for each case is treated as a random variable. It is assumed that any number of these random variables are jointly Gaussian distributed.

$$[opt_1 \quad opt_2 \quad \cdots \quad opt_n]^T \sim N(u, K) \tag{7.21}$$

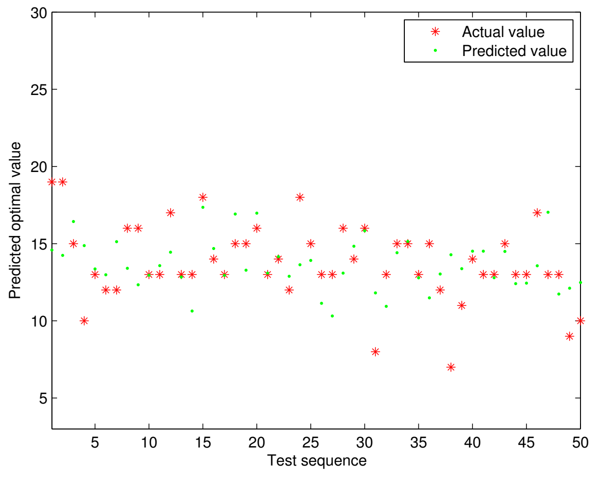


FIGURE 7.8: Predict the optimal number of parallel streams with linear regression model using 20 data points [1-20]

K is the n by n covariance matrix which is defined as the following.

$$K = \begin{bmatrix} k(X_1,X_1) & k(X_1,X_2) & \cdots & k(X_1,X_n) \\ k(X_2,X_1) & k(X_2,X_2) & \cdots & k(X_2,X_n) \\ \cdots & \cdots & \cdots & \cdots \\ k(X_n,X_1) & k(X_n,X_2) & \cdots & k(X_n,X_n) \end{bmatrix} + \sigma_n^2 I \tag{7.22}$$

$$k(X_i,X_j) = \sigma_f^2 exp\left\{\frac{-(X_i - X_j)^T(X_i - X_j)}{2l^2}\right\} \tag{7.23}$$

σ_n is the noise variance. σ_f is the maximal allowable covariance, which should be larger if the random variable covers a large range. l is the length parameter. These three parameters are named hyper-parameters which can be adjusted dynamically according to the marginal likelihood.

Based on the assumption that any number of random variables follow jointly Gaussian distribution, the predicted optimal number of parallel streams opt^* with respect to the observation X^* together with the known history data are jointly Gaussian distributed.

$$[opt_1 \quad opt_2 \quad \cdots \quad opt_n \quad opt^*]^T \sim N(u, K') \tag{7.24}$$

$$K' = \begin{bmatrix} K & K^{*T} \\ K^* & K^{**} \end{bmatrix} \tag{7.25}$$

$$K^* = [k(X_1,X^*) \quad k(X_2,X^*) \quad \cdots \quad k(X_n,X^*)] \tag{7.26}$$

$$K^{**} = k(X^*, X^*) \tag{7.27}$$

The conditional distribution of opt^* is also a Gaussian.

$$opt^* | X^*, opt_1, opt_2, \cdots, opt_n \sim N(\overline{opt^*}, \Sigma) \tag{7.28}$$

$$\overline{opt^*} = K^*(K + \sigma_n^2 I)^{-1} opt' + mean(opt) \tag{7.29}$$

$$opt' = \begin{bmatrix} opt_1 - mean(opt) \\ opt_2 - mean(opt) \\ \cdots \\ opt_n - mean(opt) \end{bmatrix} \tag{7.30}$$

$$mean(opt) = \frac{1}{n} \sum_{i=1}^{n} opt_i \tag{7.31}$$

$$\Sigma = K^{**} - K^*(k + \sigma_n^* I)^{-1} K^{*T} \tag{7.32}$$

7.3.2.2 Prediction Results

Gaussian process allows each random variable to have a separate variance. It means these random variables are not necessary to have an identical distribution. Hence it is convenient to figure out the variance for each prediction as well as the confidence region.

Similar to the linear regression model, Gaussian process can make a prediction regardless of the number of throughput measured with different number of parallel streams. The conventionality of the input X_i corresponds to the sampling times. For instance, if the throughput with respect to the number of parallel streams 1 and 2 is measured, then X_i is two dimensional.

Generally, the prediction accuracy increases with the increase of the number of throughput measured. However, the accuracy is obtained at the cost of time for sampling. Fortunately, it is possible to make a good prediction with a smaller number of sampling times with Gaussian process. Figure 7.9 shows the prediction curve on the training data set with one sampling time. Figure 7.10 shows the corresponding prediction results on the testing data set.

In Figure 7.9, the horizontal axis represents the throughput measured with 1 stream and the vertical axis represents the optimal number of parallel streams for each case. It aims to draw a curve representing the relation between the dependent variable (the optimal number of parallel streams) and the independent variable (the throughput at a certain parallelism level). The solid line represents the best estimate of the target value at the given input. The shaded area shows the 95% confidence region of the estimation. It can be seen that most of the data are covered by the 95% confidence region.

In Figure 7.10, the horizontal axis represents different testing cases and the vertical axis represents the prediction of the optimal value. The actual value, the predicted value and the 95% confidence region are all plotted.

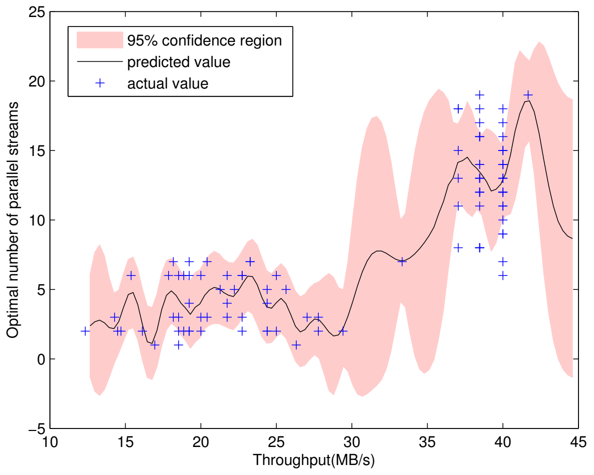


FIGURE 7.9: Train the Gaussian process model using 1 time sampling [1]

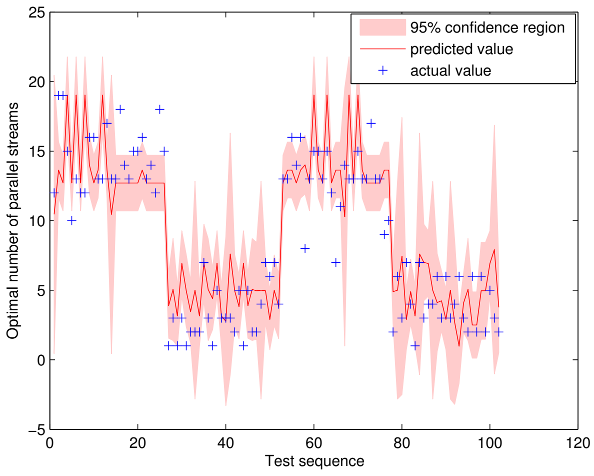


FIGURE 7.10: Predict the optimal number of parallel streams with Gaussian process using 1 data points [1]

Figure 7.11 and Figure 7.12 are two figures for the training and testing process with Gaussian process approach when using two times of sampling. In Figure 7.11, the horizontal axis represent the throughput measured when the number of parallel streams is 1 and 2 separately and the vertical axis represents the optimal number of parallel streams. It can be observed that similar throughput predicts similar optimal number of parallel streams. The convex surface and concave surface corresponds two groups of data. The data within the same group has high similarity in throughput and they have low similarity in throughput between different groups.

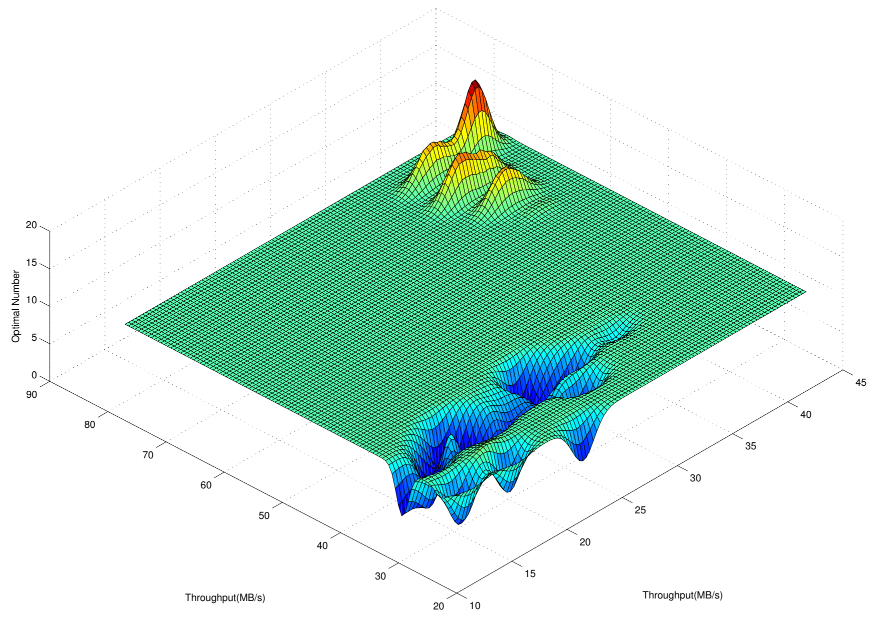


FIGURE 7.11: Train the Gaussian process model using 2 times sampling [1 2]

In this study, different approaches are introduced and compared to predict the optimal number of parallel streams. This is a significant problem in the data intensive distributed computing area since the throughput can be improved greatly with optimization. These approaches mainly falls into two categories, one of which is a pure sampling based approach and the other is the history data plus sampling approach. The full second order belongs to the first category. The disadvantage of this approach is that it does not always give an effective result. Linear regression and Gaussian process approaches belong to the second category. These two models always give an effective result and they are time efficient. The accuracy of these two models are improved compared with the pure sampling

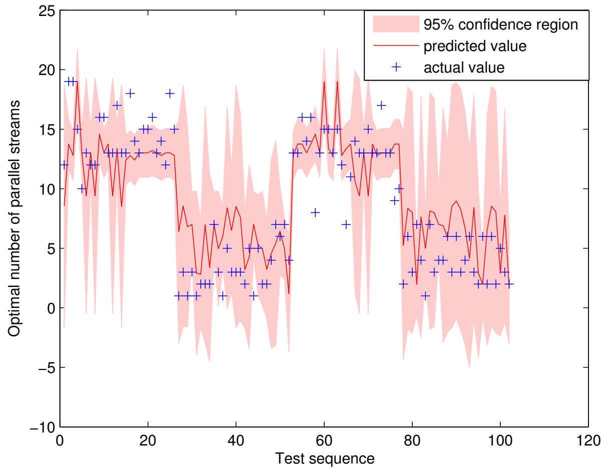


FIGURE 7.12: Predict the optimal number of parallel streams with Gaussian process using 2 data points [1 2]

based approach. Most importantly, the number of sampling times can be reduced to one or two with these two novel approaches. Hence the time cost on sampling is reduced significantly and the accuracy are guaranteed.

Chapter 8
EOS Design and Implementation for Data Scheduling

8.1 Sketch of the Optimization Service

Figure 8.1 demonstrates the structure of our design and presents two scenarios based on both GridFTP and Iperf version of the service. Site A and site B represent two machines between which the user wants to transfer data. For the GridFTP version, those machines should have GridFTP servers and GSI certificates installed. For the Iperf version, those machines should have Iperf servers running as well as a small remote module ($TranServer$) that we have implemented to perform third-party Iperf sampling. Optimization server is the orchestrator machine designated to perform the optimization of TCP parameters and store the resultant data. It also has to be recognized by the sites since the third-party sampling of throughput data will be performed by it. Client/User represents the terminal that sends out the request of optimization to the optimization server. All of them are connected via WAN or LAN.

When a user wants to transfer data between site A and site B, the user will first send a request to the optimization server, which process the request and respond to the user with the optimal parallel stream number to do the transfer. At the same time, the optimization server will estimate the optimal throughput that can be achieved and the time needed to finish the specified transfer between sites A and B. This information is also returned back to the user.

In this implementation, Stork is extended to support both estimation and optimization tasks. A task is categorized as an *estimation* task, if only estimated information regarding to the specific data movement is reported without the actual transfer. On the other hand, a task is categorized as *optimization* if the specific data movement is to be done according to the optimized estimation results. Henceforth this service is named as EOS(Estimation and Optimization Service) in short.

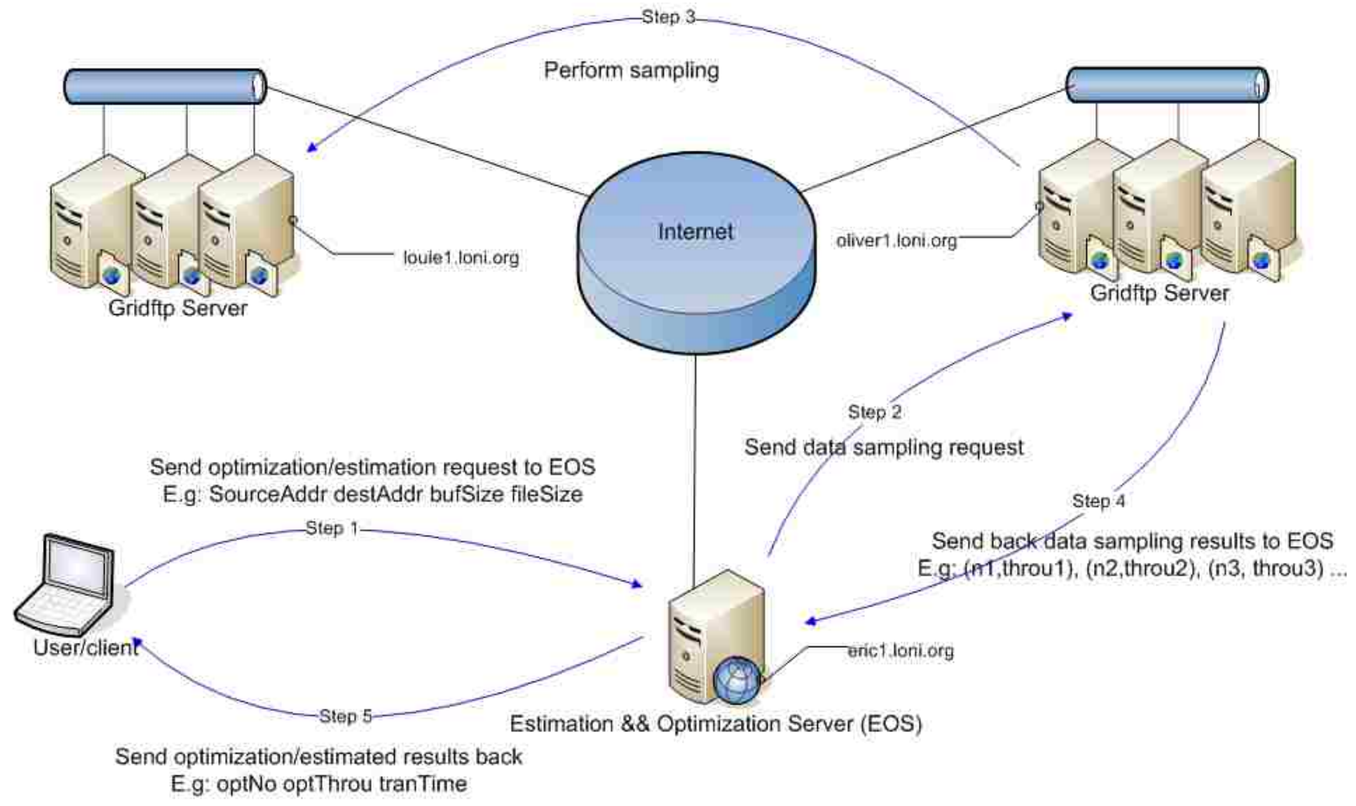

Step 3
Perform sampling
Internet
louie1.loni.org
Gridftp Server
oliver1.loni.org
Gridftp Server
Step 2
Send data sampling request
Send optimization/estimation request to EOS
E.g: SourceAddr destAddr bufSize fileSize
Step 4
Send back data sampling results to EOS
E.g: (n1,throu1), (n2,throu2), (n3, throu3) ...
Step 1
User/client
eric1.loni.org
Step 5
Estimation && Optimization Server (EOS)
Send optimization/estimated results back
E.g: optNo optThrou tranTime

FIGURE 8.1: Sketch of the optimization service

Stork inherits ClassAds structure from Condor [43] batch scheduler which are used for submission of jobs. We extend ClassAds with more fields and classify them as estimation or transfer by specifying the *dap_type* field. If it is an estimation type, it will be submitted directly to EOS, otherwise it will be submitted to the Stork server. Since an estimation task takes much shorter time than an optimization task, distinguishing the submission path by different task types enables an immediate response to the estimation tasks. *Optimization* field is added to ClassAds in order to determine if the specified transfer will adopt the optimization strategy supplied by EOS. If *optimization* is specified as YES, then the transfer is done by using the optimized parameters acquired from EOS, otherwise, it will use the default value. Another important field added to ClassAds is *use_history*. This option enforces EOS to search from the database which keeps the optimized parameters for the previous transfers of one specified source and destination pair. If there is such a record, then Stork will use the history information to perform transfers, otherwise, EOS should first perform optimization and store the information into the database, then provide Stork with the optimized parameters.

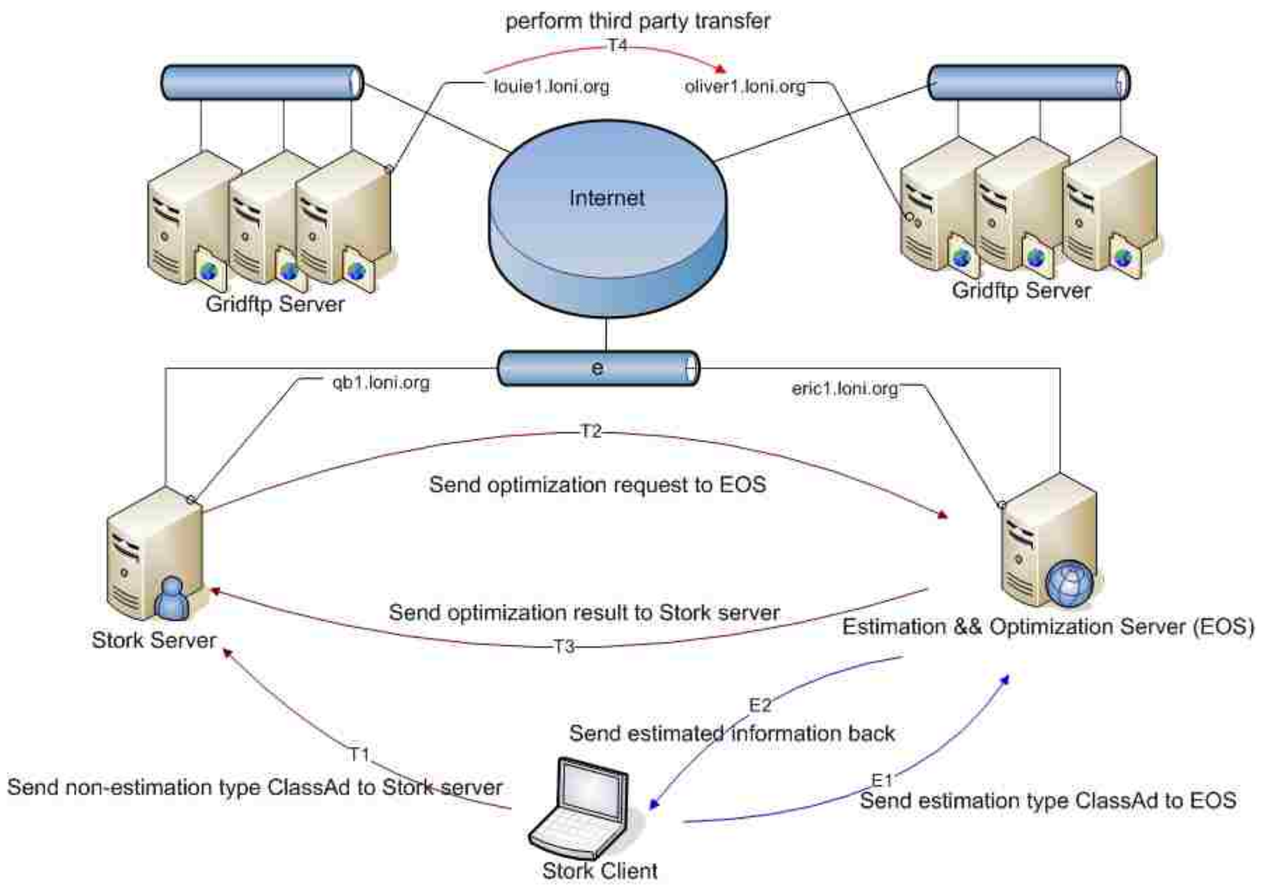

perform third party transfer
T4
louie1.loni.org
oliver1.loni.org
Internet
Gridftp Server
Gridftp Server
e
qb1.loni.org
eric1.loni.org
T2
Send optimization request to EOS
Send optimization result to Stork server
T3
Stork Server
Estimation && Optimization Server (EOS)
E2
Send estimated information back
T1
Send non-estimation type ClassAd to Stork server
E1
Send estimation type ClassAd to EOS
Stork Client

FIGURE 8.2: EOS adopted by Stork

8.2 Prediction Scheme

We have developed the full second order model to predict the aggregated throughput of parallel streams that could make accurate predictions based on only 3 samplings of different parallelism levels. The development of these models start from the foundations of the throughput equation:

$$Th_n = \frac{n}{\sqrt{p'_n}} = \frac{n}{\sqrt{a'n^2 + b'n + c'}} \tag{8.1}$$

In order to obtain the values of a', b' and c' presented in Equation 8.1, we need the throughput values of three different parallelism levels $(Th_{n_1}, Th_{n_2}, Th_{n_3})$ which can be obtained through sampling or past data transfers .

$$Th_{n_1} = \frac{n_1}{\sqrt{a'n_1^2 + b'n + c'}} \tag{8.2}$$

$$Th_{n_2} = \frac{n_2}{\sqrt{a'n_2^2 + b'n + c'}} \tag{8.3}$$

$$Th_{n_3} = \frac{n_3}{\sqrt{a'n_3^2 + b'n + c'}} \tag{8.4}$$

By solving the following three equations we could place the a',b' and c' variables to Equation 8.1 to calculate the throughput of any parallelism level.

After we calculate the optimal number of parallel streams, we can calculate the maximum throughput corresponding to that number. The optimization server needs to get at least three suitable throughput values of different parallelism levels through sampling to be able to apply the models. When the requests from the users come, the optimization server will initiate data transfers between the expected source and destination supplied by the user. This procedure terminates when the optimization server determines that it has obtained sufficient sampling data for an accurate prediction.

8.3 Implementation Technique

In this section, we present the implementation details of our service design. Depending on whether we choose to use GridFTP or Iperf, the implementation slightly differs because while GridFTP supports third-party transfers, and Iperf works as a Client/Server model. Considering the differences of the two categories of data transfer tools, we will discuss the implementation of optimization server based on both GridFTP and Iperf. The implementation technique used for these two data transfer tools can be applied to other data transfer tools no matter it supports third-party transfers or not.

The implementation of optimization service based on tools supporting third-party transfers is simply a typical Client/Server model. We have a client module running on the user site and an optimization server module running on one of the machines that is part of the Grid. On the other hand, the implementation of optimization service for data transfer tools not supporting third-party transfers such as Iperf, we need an extra module running on the remote source and destination sites to invoke the tool. The client module of the service is embedded into Stork client application and the

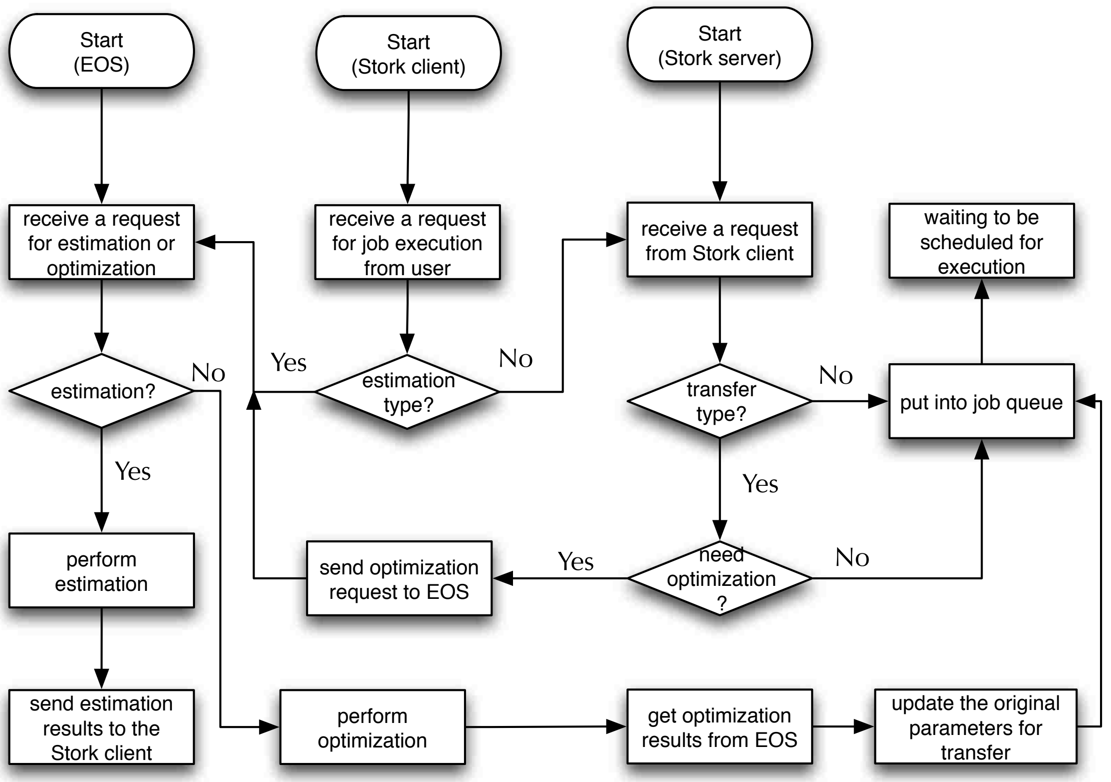

Start (EOS)
Start (Stork client)
Start (Stork server)
receive a request for estimation or optimization
receive a request for job execution from user
receive a request from Stork client
waiting to be scheduled for execution
estimation?
No
Yes
estimation type?
No
transfer type?
No
put into job queue
Yes
Yes
perform estimation
send optimization request to EOS
Yes
need optimization ?
No
send estimation results to the Stork client
perform optimization
get optimization results from EOS
update the original parameters for transfer

FIGURE 8.3: Flowchart of the EOS service

requests are done by using ClassAdds. The server module on the other hand is independent of the Stork server and able handle requests coming both from Stork client and Stork server.

Figure 8.3 shows a flowchart of the estimation and optimization module.

8.4 Optimization Server Module

The implementation of the optimization server module is more complicated than that of the client side module. The server should support multiple connections from thousands of clients simultaneously. The processing time for each client should be less than a threshold. Otherwise the user would prefer to perform the data transfer using the default configurations since the time saved by using optimized parameters cannot compensate the time waiting for the response from the optimization server.

There is a slight difference on the implementation based on tools supporting third-party transfers and those do not. In common, the optimization server keeps listening to the request from clients at a designated port. When a new request arrives, it accepts the connection and forks a child process to process that request. Then the parent process continues to listen to new connections leaving the child process to respond to the client's request.

Algorithm 8 The optimization server implementation

```
1: create a socket to be connected by the client
2: bind the socket to an empty port
3: listen to this port
4: while TRUE do
5:    if a new connection request arrives then
6:       the optimization server accepts the connection from the client program
7:       $processId \leftarrow fork()$
8:       if $processId = parent\ processId$ then
9:          back to listening to the designated port
10:      else {$processId = children\ processId$}
11:         receive the request information from the client
12:         perform sampling transfers
13:         build a mathematical model and process the sampling results
14:         send back the optimized parameters to the clients
15:      end if
16:   else {no new connection request comes}
17:      block until a new connection comes
18:   end if
19: end while
```

The child process is responsible for sampling data transfers between the remote sites and get the data pairs (throughput and number of parallel streams) from them. Then it will analyze the data and generate an aggregate throughput function with respect to the number of parallel streams. Finally it will calculate the maximum aggregate throughput with respect to the optimal number of parallel streams and send back the information to the client. Algorithm 8 presents the outline of the optimization server.

At step 13 in Algorithm 8, the performing of sampling transfers is different on data transfer tools that support third-party transfers and tools that does not support third-party transfers. For the implementation based on GridFTP, the child process is able to invoke *globus-url-copy* command to control the data transfers between the remote sites. However, for the implementation based on Iperf, the child process belonging to the optimization server has no privilege to control the data transfers between the remote sites. We need an extra module running on the remote sites that can be connected by the optimization server. So the optimization server plays dual roles. When a request comes from the client it acts as a server and when it asks the remote module to start Iperf transfers it acts as a client.

8.5 Quantity Control of Sampling Data Transfers

The time interval between the arrival of a request from the client and an optimized decision made for the corresponding request mainly depends on the time consumed on the sampling data transfers. The cost of application of the mathematical model on the sampling data and derivation of optimal parameters is negligible, around several milliseconds on a $2.4Ghz$ CPU. However, each sampling data transfer takes nearly 1 second based on the sampling size. At least 3 sampling data transfers are required because of the property of the mathematical model we propose. However relying only on 3 measurements makes the models susceptible to the correct selection of the three parallelism levels.

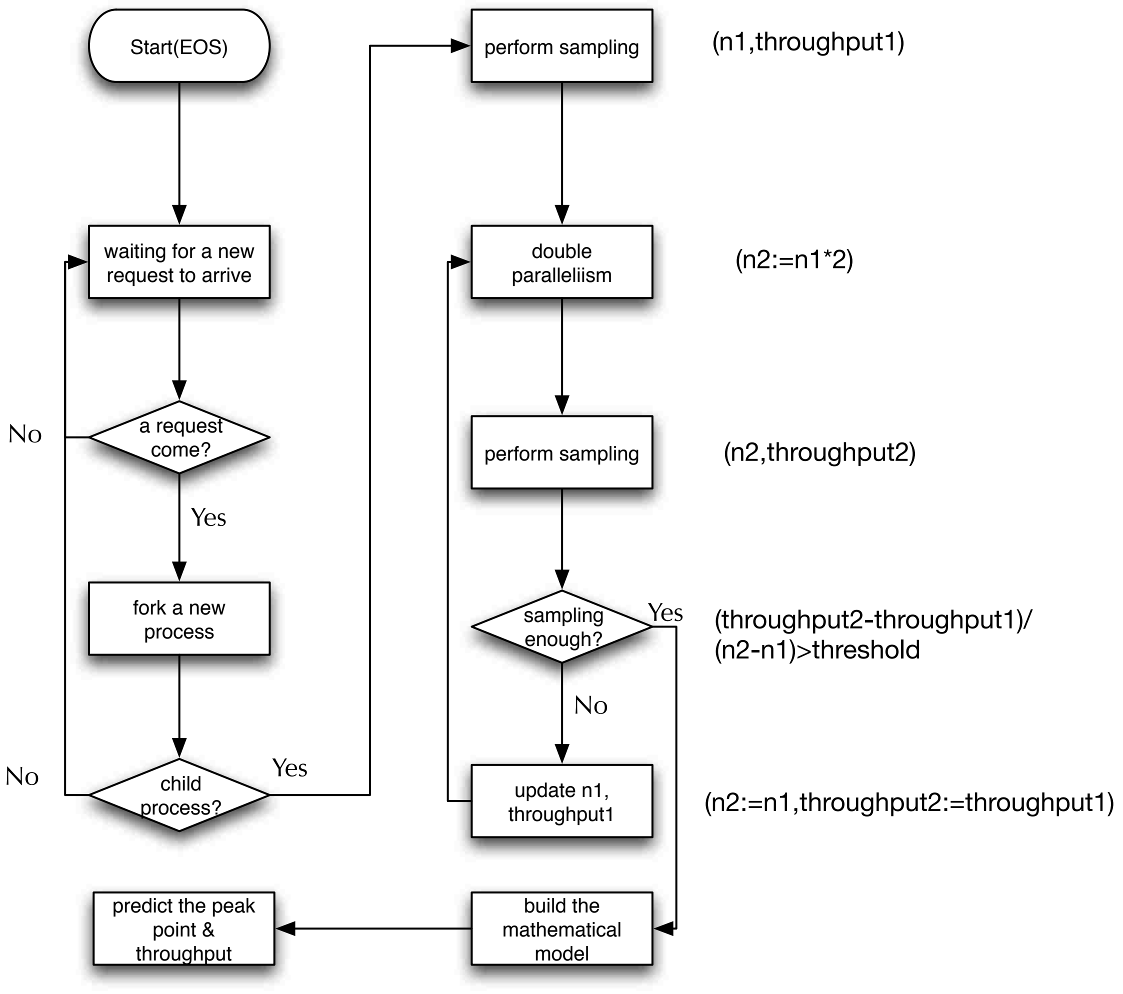


FIGURE 8.4: EOS sampling

We propose to find a solution to satisfy both the time limitation and the accuracy requirements. Our approach doubles the number of parallel streams for every iteration of sampling, and observe the corresponding throughput. While the throughput increases, if the slope of the curve is below a threshold between successive iterations, the sampling stops. Another stopping condition is if the throughput decreases compared to the previous iteration before reaching that threshold. Figure 8.4 illustrates the role of sampling plays in EOS.

Let n be the number of parallel streams with respect to the maximum aggregated throughput of the underlying network. According to our exponentially increasing scheme,the total sampling time s is equal to the logarithm of n, i.e, $s = \log n$. For example, if the optimal parallel number of streams is less than 32, we only need less than 5 sampling iterations.

8.6 Scheduling of EOS

Generally speaking, an estimation type task costs less time than an optimization type task as the former does not need to transfer the whole data from the source to the destination. Taking the variety of time consumption into consideration, the estimation type task is submitted to the EOS directly and the optimization type task is firstly submitted to the Stork server and then submitted to EOS by the Stork scheduler. This makes sense since the shortest task is expected to finish as early as possible. A strict shortest task first strategy guarantees that the total waiting time is minimized.

A triple of source, destination and arguments is introduced to characterize each individual task.

$$Triple(task_i) =< S_i, D_i, A_i > \tag{8.5}$$

S, D, and A represent source, destination and arguments separately. $Triple(task_i)$ is said to be equivalent to $Triple(task_j)$ if they have the same source, destination pairs as well as the arguments.

$$\begin{gathered} Triple(task_i) = Triple(task_j) \\ \Longleftrightarrow \\ (S_i = S_j \ \wedge \ D_i = D_j \ \wedge \ A_i = A_j) \\ \vee \ (S_i = D_j \ \wedge \ D_i = S_j \ \wedge \ A_i = A_j) \end{gathered} \tag{8.6}$$

Two tasks are said to be identical if the *Triple* operations on them are equivalent, and they are said to be orthogonal if none of the elements in the triple are identical.

$$task_i = task_j \Longleftrightarrow Triple(task_i) = Triple(task_j) \tag{8.7}$$

$$\begin{aligned} task_i \perp task_j \Longleftrightarrow & (S_i \neq S_j \ \wedge \ D_i \neq D_j \ \wedge \ A_i \neq A_j) \\ & \wedge (S_i \neq D_j \ \wedge \ D_i \neq S_j \ \wedge \ A_i \neq A_j) \end{aligned} \tag{8.8}$$

Furthermore, two tasks are said to be similar if they have one element in common in the source and destination pairs, and they are said to be approximate to each other if their source and destination pairs are identical while the arguments not.

$$\begin{aligned} task_i \sim task_j \Longleftrightarrow & (S_i \neq S_j \ \wedge \ D_i = D_j) \\ & \vee (S_i = S_j \ \wedge \ D_i \neq D_j) \\ & \vee (S_i = D_j \ \wedge \ D_i \neq S_j) \\ & \vee (S_i \neq D_j \ \wedge \ D_i = S_j) \end{aligned} \tag{8.9}$$

$$\begin{aligned} task_i \approx task_j \Longleftrightarrow & (S_i = S_j \ \wedge \ D_i = D_j \ \wedge \ A_i \neq A_j) \\ & \vee (S_i = D_j \ \wedge \ D_i = S_j \ \wedge \ A_i \neq A_j) \end{aligned} \tag{8.10}$$

Introduce a term $cor(task_i, task_j)$ to denote the correlation score between two tasks.

$$cor(task_i, task_j) = \begin{cases} 0 & task_i \perp task_j \\ \alpha & task_i \sim task_j \\ \beta & task_i \approx task_j \\ 1 & task_i = task_j \end{cases} \tag{8.11}$$

In Equation 8.11 α and β variate between systems. They satisfy the following constrains.

$$0 \leq \alpha \leq \beta \leq 1 \tag{8.12}$$

If two tasks have the same source and destination pairs then their correlation score will be β or 1. In other words, these two tasks have the equivalent to or approximate to relationship, indicating that they are closely related to each other. Tasks having such relationship are not allowed to be executed parallel since they will affect each other. Actually, if two tasks have the equivalent to relationship, then these two tasks have the same optimized parameters. Only one of them needs to be executed and then the results are sent to both of them. If two tasks have the approximate to relationship, then these two tasks should be done sequentially, otherwise the measured sampling throughput of two tasks will be less if they are allowed to be executed concurrently. The inaccuracy of the throughput has significant impact on the mathematical model which the estimation and optimization relay on.

On the contrary, if two tasks have different source and destination pairs, their correlation score will be 0. And if they have one in common in the source and destination pairs, their correlation score is identified as α. Definitely, two tasks are allowed to be executed parallel if their correlation score is zero. However, it is hard to make a decision when their correlation score is α. In a low throughput network,the network interface card (NIC) of 1Gbps is capable to handle hundreds of links concurrently without affecting each other. However, in a high throughput network such as 10Gbps, it will be a bottleneck. In this situation, α approximates to 0. Even if there are only two links connected, they will affect each other's transferring rate. In this situation, α approximates to 1.

The tasks that can be executed parallel should be maximized provided the system is not overloaded. The maximal number of parallel tasks can be configured in the configure file of EOS. Meanwhile, each task should be as representative as possible. A task is said to be representative to another task if they are equivalent to each other. The more equivalent tasks it has the more representative it is.

$$equ(task_i, task_j) = \begin{cases} 1 \, task_i = task_j \\ 0 \, otherwise \end{cases} \tag{8.13}$$

$$rep(task_1, task_2, ..., task_n) = \sum_{i=2}^{n} equ(task_1, task_i) \tag{8.14}$$

Introduce a term confusion score for the parallel tasks.

$$conf(task_1, task_2, ..., task_k) = \sum_{i=1}^{k} \sum_{j=i+1}^{k} cor(task_i, task_j) \tag{8.15}$$

Each source or destination represents a host address. Also introduce a penalty term for each host.

$$ind(host, task_i) = \begin{cases} 0 \, host \neq S_i \ \wedge \ host \neq D_i \\ 1 \, host = S_i \ \vee \ host = D_i \end{cases} \tag{8.16}$$

$$penal(host) = \sum_{i=1}^{k} \alpha * ind(host, task_i) \tag{8.17}$$

The penalty term indicates the frequency of one specified host appears in the source and destination pairs of the parallel tasks list. For instance, if the tasks list is $< h1, h2, a1 >, < h1, h3, a2 >, < h2, h4, a3 >$, then $penal(h1) = penal(h2) = 2 * \alpha$, and $penal(h3) = penal(h4) = \alpha$. A threshold for the penalty term, namely $\bigtriangledown$, is defined as the upper bound for each host appears in the k parallel tasks. The penalty for each host should satisfy the following condition.

$$penal(host_i) \leq \bigtriangledown \tag{8.18}$$

Suppose the maximal number of parallel tasks allowed to be executed concurrently by EOS is n. There are s possible hosts that appear in these source and destination pairs. To get the optimal solution for the EOS scheduling problem the number of parallel tasks should be as large as possible. Meanwhile, the confusion score should be as small as possible and the constrain on the penalty term of each host should be satisfied. Specifically,

$$\begin{aligned}(\hat{k}; \widehat{task_i}) &= \operatorname*{argmax}_{k:k\leq n} \min_{task_i:1,...,k} conf(task_1, task_2, ..., task_k) \\ subject\ to: &\qquad penal(host_j) \leq \bigtriangledown \quad \forall j:\ 1 \leq j \leq s\end{aligned} \tag{8.19}$$

To simplify the discussion, we use a sparse matrix to represent the tasks received by EOS at a given time T. The first column vector consists of all the candidate parallel tasks. The tasks in each line vector are not allowed to be executed parallel.

$$task = \begin{bmatrix} task_{10} & task_{11} & task_{12} & ... & task_{1t_1} \\ task_{20} & task_{21} & task_{22} & ... & task_{2t_2} \\ ... & ... & ... & ... & ... \\ task_{N0} & task_{N1} & task_{N2} & ... & task_{Nt_N} \end{bmatrix}$$

When a new task is submitted to EOS, the first column will be searched. If there exists one task in the column vector such that it has the same source and destination pairs, then the new task will be appended to the end of the corresponding line vector. Otherwise, the new task will be appended to the end of the column vector. A brief description of how to add a new task is

Algorithm 9 AddTask($newtask$)

```
visit the column vector $[task_{10}, task_{20}, ..., task_{N0}]^T$
if $\exists i$ such that $cor(task_{i0}, newtask) \in \{1, \beta\}$ then
    append $newtask$ to the end of the line vector,
    $[task_{i0}, ..., task_{it_i}] \leftarrow [task_{i0}, ..., task_{it_i}, newtask]$
else
    append $newtask$ to the end of the column vector,
    $[task_{10}, ..., task_{N0}]^T \leftarrow [task_{10}, ..., task_{N0}, newtask]^T$
end if
```

Algorithm 10 DeterminParallelTasks-1($task, n, \alpha, \beta, \bigtriangledown$)

```
$i \leftarrow 1$
initialize the penalty term for each host to be 0 :
$penal(host) \leftarrow 0$
while $i \leq N \ \wedge \ |task_{par}| < n$ do
    if $penal(source_{i0}) < \bigtriangledown \ \wedge \ penal(destination_{i0}) < \bigtriangledown$ then
        append $task_{i0}$ to the end of $task_{par}$
        $penal(source_{i0}) \leftarrow penal(source_{i0}) + \alpha$
        $penal(destination_{i0}) \leftarrow penal(destination_{i0}) + \alpha$
    end if
    $i \leftarrow i + 1$
end while
```

shown in Algorithm 9. It is easy to verify that: for the first column vector: $\forall i, j \in \{1, 2, .., N\}$, we have $cor(task_{i0}, task_{j0}) \in \{0, \alpha\}$. For any given line vector, e.g $[task_{k0}, task_{k1}, ..., task_{kt_k}]$, $\forall i, j \in \{0, 1, 2, ..., t_k\}$, we have $cor(task_{ki}, task_{kj}) \in \{1, \beta\}$.

The first column vector contains all the candidate tasks that can be executed parallel. Hence the most efficient way to determine the parallel tasks is to traverse this vector. If adding one task from this vector into the parallel tasks vector, the penalty constrain is not violated, then this task is added to the parallel tasks vector. A brief description of this method is shown in Algorithm 10. The time complicity of this algorithm is $O(N)$ since each of the tasks in the column vector will be visited in the worst case.

Algorithm 10 is time-efficient. However, it is not optimal since the confusion score of the parallel tasks determined by this algorithm is not minimal. There might exist a parallel tasks vector such that its confusion score is zero, but this algorithm gives a solution that has a larger confusion score. In order to reduce the confusion score, an improved solution is shown in Algorithm 11. In this solution, the first column vector is traversed to extract all the possible tasks such that they are mutually

Algorithm 11 DeterminParallelTasks-2($task, n, \alpha, \beta, \bigtriangledown$)

```
initialize the parallel tasks to be an empty vector: $task_{par} \leftarrow [\ ]$
$i \leftarrow 1$
while $i \leq N \ \wedge \ |task_{par}| < n$ do
    $j \leftarrow 1$
    while $j \leq |task_{par}|$ do
        if $cor(task_{par}(j), task_{i0}) \neq 0$ then
            break;
        else
            $j \leftarrow j + 1$
        end if
    end while
    if $j = |task_{par}| + 1$ then
        append $task_{i0}$ to the end of $task_{par}$
        mark $task_{i0}$ as selected
        $penal(source_{i0}) \leftarrow \alpha$
        $penal(destination_{i0}) \leftarrow \alpha$
    end if
    $i \leftarrow i + 1$
end while
$i \leftarrow 1$
if $|task_{par}| < n$ then
    while $i \leq N \ \wedge \ |task_{par}| < n$ do
        if $task_{i0}$ is not selected $\wedge$
        $penal(source_{i0}) < \bigtriangledown \ \wedge$
        $pelal(destination_{i0}) < \bigtriangledown$ then
            append $task_{i0}$ to the end of $task_{par}$
            $penal(source_{i0}) \leftarrow penal(source_{i0}) + \alpha$
            $penal(destination_{i0}) \leftarrow penal(destination_{i0}) + \alpha$
        end if
        $i \leftarrow i + 1$
    end while
end if
```

Algorithm 12 ProcessParallelTasks($task_{par}$)

```
create a new thread for each task in $task_{par}$
for each $task_{par}(i) \in task_{par}$
suppose $task_{par}(i)$ maps to $task_{k0}$ in $task$
$j \leftarrow 0$
while $j \leq t_j$ do
    if $cor(task_{k0}, task_{kj}) = 0$ then
        send the optimized parameters to the owner of $task_{kj}$
        remove $task_{kj}$ from the line vector:
        $[task_{k0}, task_{k1}, ..., task_{kt_j}]^T$
    end if
    $j \leftarrow j + 1$
end while
```

orthogonal. After this step the confusion score of the parallel tasks vector is zero. If the size of this vector has reached its upper bound n, then it is done. Otherwise, traverse the first column vector a second time to add more tasks into the parallel tasks vector. The time complicity for this algorithm is $O(n^2 + N)$.

After the parallel tasks vector is set, then a new thread is created for each task in this vector. All of them will be executed concurrently. For each of them, when it is finished, it should find the corresponding entry in the first column vector and return the results back to all the tasks that have the same arguments in that line vector and remove them from this vector. A brief description is illustrated in Algorithm 12.

Define the performance gain as the ratio of the number of tasks done with a representative strategy to that without it within a time unit. For instance, suppose the parallel tasks vector happens to be the first column vector of $task$.

$$task_{par} = [task_{10}, task_{20}, ..., task_{N0}]^T$$

The gain can be calculated as the following.

$$gain(task) = \frac{N + \sum_{i=1}^{N} rep(task_{i0}, task_{i1}, ..., task_{it_i})}{N} \tag{8.20}$$

If most of the tasks in the line vectors have the same arguments, then we have:

$$conf(task_{k0}, task_{k1}, ..., task_{kt_k}) \propto \frac{t_k * (t_k + 1)}{2} \tag{8.21}$$

$$gain(task) \propto \frac{N + \sum_{i=1}^{N} t_i}{N} \tag{8.22}$$

On the contrary, if most of the tasks in the line vectors have different arguments, then we have:

$$conf(task_{k0}, task_{k1}, ..., task_{kt_k}) \propto \frac{t_k * (t_k + 1) * \beta}{2} \tag{8.23}$$

$$gain(task) \propto \frac{N}{N} = 1 \tag{8.24}$$

Fortunately, it turns out that the first case holds in most time since the users often use the default arguments set by the system. Hence the performance can be improved significantly. Furthermore, research is being done to deduce the optimal parameters according to the hidden rules between these different parameters. Then it is possible to improve the gain significantly even in the second case.

In the following, a simple example illustrates how the above algorithms work. T is the task matrix at a given time. T_{par} represents the parallel tasks vector. T_1 represents the residual tasks after one round scheduling. Suppose the maximal number of parallel tasks allowed by EOS is 3, and the threshold is $\bigtriangledown = 2\alpha$.

$$T = \begin{bmatrix} <1,2,1><1,2,2><1,2,1><1,2,1> \\ <3,4,1><3,4,1><3,4,1><3,4,1> \\ <1,3,1><1,3,2><1,3,2><1,3,1> \\ <5,6,1><5,6,2><5,6,1><5,6,2> \end{bmatrix}$$

Apply Algorithm 10, the following can be obtained:

$$T_{par} = [<1,2,1>, <3,4,1>, <1,3,1>]^T$$

$$T_1 = \begin{bmatrix} <1,2,2> \\ <1,3,2><1,3,2> \\ <5,6,1><5,6,2><5,6,1><5,6,2> \end{bmatrix}$$

$$gain(T) = \frac{3 + (2 + 3 + 1)}{3} = 3$$

Apply Algorithm 11, the following can be obtained:

$$T_{par} = [<1,2,1>, <3,4,1>, <5,6,1>]^T$$

$$T_1 = \begin{bmatrix} <1,2,2> \\ <1,3,1><1,3,2><1,3,2><1,3,1> \\ <5,6,2><5,6,2> \end{bmatrix}$$

$$gain(T) = \frac{3 + (2 + 3 + 1)}{3} = 3$$

8.7 The Framework Applied in Stork

Condor, which was developed in University of Wisconsin, works perfectly in Grid network environment. Users submit classAds to the condor server, and then the condor server process it. Stork is a data scheduler in Condor which schedules and processes the classAds if they are transfer types. The Stork server use the default settings of TCP. Hence an estimation and optimization service (EOS) is deployed to assist Stork server and Condor server to achieve high throughput and performance. Figure 8.2 illustrates the cooperation of Stork, and EOS. The Stork user simply indicates that whether or not they will choose the optimization options in the job submission file (i.e. in a *classAd*), and the Stork server will communicate with the EOS transparently. Since the Stork server puts all the incoming requests into a job queue and usually there are many jobs, it will take a long time to get the response from Stork server, which is not reasonable. Alternatively, the user can simply send estimation and optimization request directly to the EOS since we have implemented the interface on the Stork client side.

8.8 Experimental Results

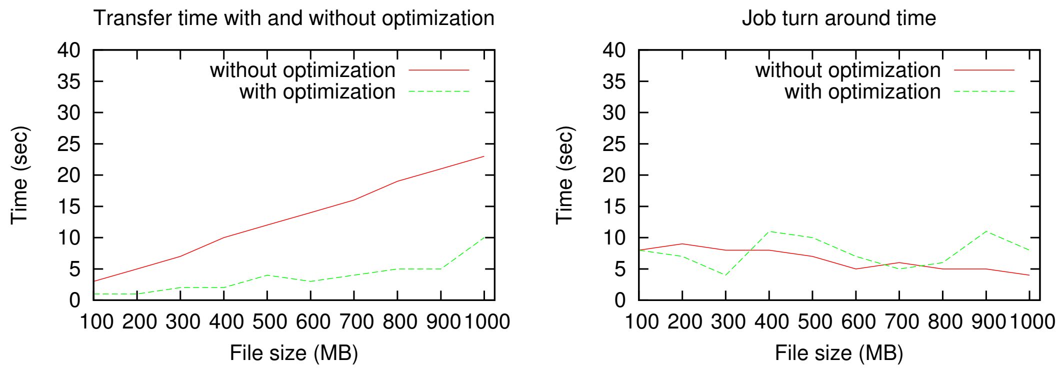


FIGURE 8.5: Data transfer time and job turn around time from Eric to Oliver

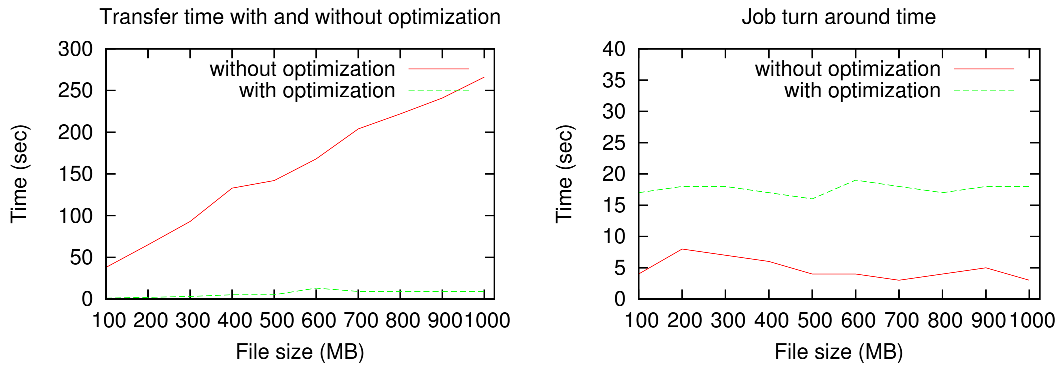


FIGURE 8.6: Data transfer time and job turn around time from Painter to louie

The current implementation of EOS applies the pure sampling based approach. Experiments show that the throughput can be improved significantly with optimization approaches in EOS. Figure 8.5 represents the data transfer time and job turn around time between two LONI clusters (eric

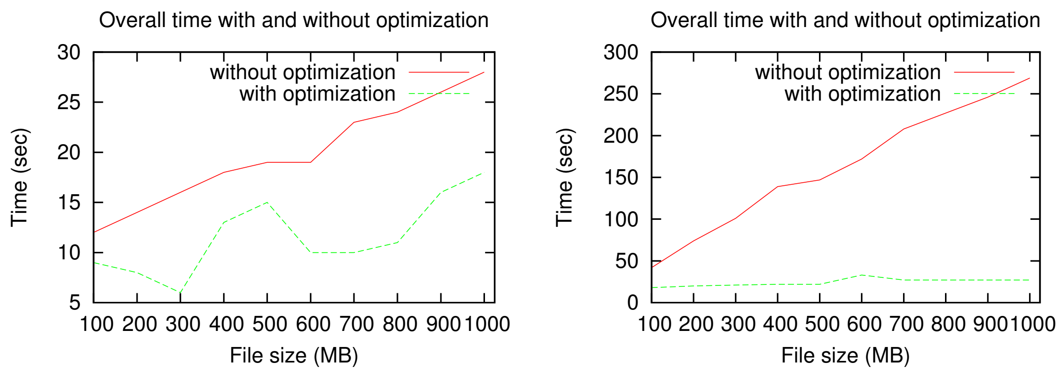


FIGURE 8.7: Overall time comparison

and olvier). Figure 8.6 represents the measurement between another two LONI clusters (painter and louie). The job turnaround time consists of the job waiting time and the throughput sampling time when optimization is applied. It can be seen that the sampling time the bottleneck for optimization especially when the number of sampling times is large (painter to louie). Figure 8.7 represents the overall time for data transfers with and without optimization. The optimized approach outperforms the non-optimized one even though the sampling time is nontrivial.

Chapter 9
Conclusions

This research proposed a data-aware scheduling approach for the scientific data intensive workflow. Different from the existing workflow scheduling algorithms, the proposed scheduling algorithm is data-aware, which means the scheduling algorithm will consider all the data involved in the workflow, for instance, the data to be staged in from remote data centers, the intermediate data generated by the tasks in the workflow and the data to be staged out to the remote data centers. The proposed data-aware scheduling algorithm allows data staging to participate in the scheduling process, which will prevent a biased scheduling without considering data movement. The data-aware scheduling is based on A-star algorithm and the most challenging problem is to design a efficient heuristics function. On the other hand, since the workflow is data intensive, there will be large amount of data moving between the computation sites during the execution of the workflow. Hence it is important to guarantee the data to be delivered in an efficient manner. Also an approximate scheduling algorithm based on genetic algorithm (GA) is proposed to address the data-aware workflow scheduling problem. Data transfer rate directly affects the turnaround time of the whole workflow. It is crucial to deliver the data as fast as possible in the workflow. To achieve this goal, several throughput optimization models are proposal, compared and improved in this research. The major contributions made in this research consists of the following aspects:

Improve the existing workflow scheduling algorithm

The existing workflow scheduling algorithms assume that task execution and data transfer are in sequential. Based on these algorithms, when one task is transferring intermediate data to other computation site from its execution site, the execution site is not allowed to execute other ready tasks even the processor is free. The scheduling derived by these algorithms is not optimal especially when the intermediate data amount are large. Results show that by overlapping the intermediate data transfer and task execution, the turnaround time of the data intensive workflow can be reduced, moreover, the algorithm is time efficient.

Design and implement a data-aware scheduling algorithm

The existing optimal workflow scheduling algorithm only considers the intermediate data. In practical applications, it is quite often that tasks in a data-intensive workflow need data to be staged in from remote data centers for computation and will stage out some intermediate data to the remote data centers for future use. With the existing workflow algorithm, the external data are expected to be staged in first before the workflow execution, and the data of interest will be staged out after the execution of the whole workflow. The scheduling process are divided into three phases, which couldn't guarantee a global optimal solution. In the proposed data-aware scheduling, the mapping from a task to a processor is decided by all these factors matters, for instance, data to be staged in, data to be staged out, intermediate data, network and computation power.

In order to evaluate the data-aware scheduling, we re-implement one of the best workflow scheduling algorithms as a benchmark. Both homogeneous and heterogeneous distributed systems are evaluated. Results show that the proposed data-aware scheduling algorithm indeed outperforms the existing best scheduling algorithm in terms of turnaround time and efficiency.

Also the approach to find an approximate scheduling is proposed based on genetic algorithm. The proposed GA based algorithm differs from the previous research in two significant ways. First, it is data-aware. The fitness function considers all the data involved in the workflow, the stage in data, stage out data, and the intermediate data. Second, the mutation and crossover operation are simple but effective. Experiment results show that the GA based approach can give a scheduling close to the optimal one.

Improve the existing throughput prediction model

To address the data-aware scheduling problem, a high throughput for data delivery in the distributed system is essential. However, the current throughput prediction models are not accurate and not suitable for the throughput optimization problem. Inspired by an existing model, a new model is proposed and tested. Experiments show that the proposed model improves the accuracy of throughput prediction.

In order to choose the best suitable model for throughput optimization, a comprehensive comparison has been done over the existing throughput prediction models. Experiments show that the improved throughput prediction model outperforms the other model both in accuracy and stability. The improved throughput prediction model is able to sustain noise data to a certain degree while other models are prone to make wrong predictions when noise data are presented.

Design and implement the estimation and optimization service (EOS)

The estimation and optimization service are designed to predict the optimal number of parallel streams when the TCP throughput achieves peak performance and becomes stable. Given a source and destination pairs of the data, EOS will perform data sampling varying the number of parallel streams according a sophisticated sampling policy. Then the sampling data will be passed to the mathematic model with three parameters. An algorithm is designed to figure out which sampling data will give the best prediction model. Finally, the optimal number of parallel streams is derived from the prediction model as well as the corresponding throughput.

References

[1] Amazon elastic compute cloud. http://aws.amazon.com/ec2.

[2] Energy sciences network (ESNet). http://www.es.net/.

[3] Louisiana optical network initiative (LONI). http://www.loni.org/.

[4] TeraGrid. http://www.teragrid.org/.

[5] Windows azure. http://www.microsoft.com/windowsazure.

[6] William Allcock, John Bresnahan, Rajkumar Kettimuthu, Michael Link, Catalin Dumitrescu, Ioan Raicu, and Ian Foster. The globus striped gridftp framework and server. In *SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, page 54, Washington, DC, USA, 2005. IEEE Computer Society.

[7] E. Altman, D. Barman, B. Tuffin, and M. Vojnovic. Parallel tcp sockets: Simple model, throughput and validation. pages 1–12, April 2006.

[8] H. Balakrishman, V. N. Padmanabhan, S. Seshan, and R. H. Katz M. Stemm. Tcp behavior of a busy internet server: Analysis and improvements. pages 252–262, California, USA, March 1998.

[9] Pei Chann Chang and Yen Shean Jiang. A state-space search approach for parallel processor scheduling problems with arbitrary precedence relations. *European Journal of Operational Research*, 77:208–223, 1994.

[10] Hong Chich Chou and Chung-Ping Chung. Optimal multiprocessor task scheduling using dominance and equivalence relations. *Computers & OR*, 21(4):463–475, 1994.

[11] J. Crowcroft and P. Oechslin. Differentiated end-to-end internet services using a weighted proportional fair sharing tcp. *ACM SIGCOMM Computer Communication Review*, 28(3):53–69, July 1998.

[12] E. Yildirim D. Yin and T. Kosar. A data throughput prediction and optimization service for widely distributed many-task computing. *Super computing-MTAGS*, 2011.

[13] S. Kulasekaran B. Ross D. Yin, E. Yildirim and T. Kosar. A data throughput prediction and optimization service for widely distributed many-task computing. *To appear in IEEE Transactions on Parallel and Distributed Systems*, 2011.

[14] Ewa Deelman, James Blythe, A Gil, Carl Kesselman, Gaurang Mehta, Sonal Patil, Mei hui Su, Karan Vahi, and Miron Livny. Pegasus: Mapping scientific workflows onto the grid. pages 11–20, 2004.

[15] L. Eggert, J. Heideman, and J. Touch. Effects of ensemble tcp. *ACM Computer Communication Review*, 30(1):15–29, 2000.

[16] S. Floyd. Rfc3649: Highspeed tcp for large congestion windows.

[17] Thilina Gunarathne, Tak-Lon Wu, Judy Qiu, and Geoffrey Fox. Cloud computing paradigms for pleasingly parallel biomedical applications. In *HPDC*, pages 460–469, 2010.

[18] T. J. Hacker, B. D. Noble, and B. D. Atley. The end-to-end performance effects of parallel tcp sockets on a lossy wide area network. pages 434–443, 2002.

[19] T. J. Hacker, B. D. Noble, and B. D. Atley. Adaptive data block scheduling for parallel streams. pages 265–275, July 2005.

[20] Christina Hoffa, Gaurang Mehta, Tim Freeman, Ewa Deelman, Kate Keahey, Bruce Berriman, and John Good. On the Use of Cloud Computing for Scientific Workflows. In *ESCIENCE '08: Proceedings of the 2008 Fourth IEEE International Conference on eScience*, Washington, DC, USA, 2008. IEEE Computer Society.

[21] Edwin S. H. Hou, Nirwan Ansari, and Hong Ren. A genetic algorithm for multiprocessor scheduling. *IEEE Trans. Parallel Distrib. Syst.*, 5(2):113–120, 1994.

[22] Yong Zhao Ioan Raicu, Ian Foster. Many-task computing for grids and supercomputers.

[23] Manish Jain and Constantinos Dovrolis. Pathload: A measurement tool for end-to-end available bandwidth. In *In Proceedings of Passive and Active Measurements (PAM) Workshop*, pages 14–25, 2002.

[24] Cheng Jin, D. X. Wei, Steven H. Low, G. Buhrmaster, J. Bunn, D. H. Choe, R. L. A. Cottrell, J. C. Doyle, W. Feng, O. Martin, H. Newman, F. Paganini, S. Ravot, and S. Singh. Fast tcp: from theory to experiments. *IEEE Network*, 19(1):4–11, February 2005.

[25] R. P. Karrer, J. Park, and J. Kim. Adaptive data block scheduling for parallel streams. Technical report, Deutsche Telekom Labs, 2006.

[26] R. Kelly. Scalable tcp: Improving performance in highspeed wide area networks. *Computer Communication Review*, 32(2), 2003.

[27] G. Kola and M. K. Vernon. Target bandwidth sharing using endhost measures. *Performance Evaluation*, 64(9-12):948–964, October 2007.

[28] Yu-Kwong Kwok and Ishfaq Ahmad. On multiprocessor task scheduling using efficient state space search approaches. *J. Parallel Distrib. Comput.*, 65(12):1515–1532, 2005.

[29] J. Lee, D. Gunter, B. Tierney, B. Allcock, J. Bester, J. Bresnahan, and S. Tuecke. Applied techniques for high bandwidth data transfers across wide area networks. Beijing, China, September 2001.

[30] Jin-Cherng Lin. Optimal task assignment with precedence in distributed computing systems. *Inf. Sci.*, 78(1-2):1–18, 1994.

[31] D. Lu, Y. Qiao, and P. A. Dinda. Characterizing and predicting tcp throughput on the wide area network. pages 414–424, 2005.

[32] D. Lu, Y. Qiao, P. A. Dinda, and F. E. Bustamante. Modeling and taming parallel tcp on the wide area network. page 68b, April 2005.

[33] Bertram Ludascher, Ilkay Altintas, Chad Berkley, Dan Higgins, Efrat Jaeger, Matthew Jones, Edward A. Lee, Jing Tao, and Yang Zhao. Scientific workflow management and the kepler system. In *Concurr. Comput. : Pract. Exper*, page 2006, 2005.

[34] Haiyang Wang Yanbing Bi Ji Bian Meng xu, Lizhen Cui. A data-intensive workflow scheduling algorithm for grid computing.

[35] Srikuma Venugopal Mustafizur Rahman and Rajkumar Buyya. A dynamic critical path algorithm for scheduling scientific workflow applications on global grids. *Proc. Third IEEE International Conference on e-Science and Grid Computing*, pages 35–42, 2007.

[36] Wei neng Chen, Student Member, Jun Zhang, and Senior Member. An ant colony optimization approach to a grid workflow scheduling problem with various qos requirements.

[37] Tom Oinn, Matthew Addis, Justin Ferris, Darren Marvin, Tim Carver, Matthew R. Pocock, and Anil Wipat. Taverna: A tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20:2004, 2004.

[38] Rob Pennington. Terascale clusters and the teragrid. In *Proceedings for HPC Asia*, pages 407–413. TeX Users Group, Dec 2002.

[39] Pascale Primet, Robert Harakaly, and Franck Bonnassieux. Experiments of network throughput measurement and forecasting using the network weather. In *CCGRID '02: Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, page 413, Washington, DC, USA, 2002. IEEE Computer Society.

[40] Vinay J. Ribeiro, Rudolf H. Riedi, Richard G. Baraniuk, Jiri Navratil, and Les Cottrell. pathchirp: Efficient available bandwidth estimation for network. In *In Passive and Active Measurement Workshop*, 2003.

[41] H. Sivakumar, S. Bailey, and R. L. Grossman. Psockets: The case for application-level network striping for data intensive applications using high speed wide area networks. pages 63–63, Texas, USA, November 2000.

[42] Jacob Strauss, Dina Katabi, and M. Frans Kaashoek. A measurement study of available bandwidth estimation tools. In *Internet Measurement Comference*, pages 39–44, 2003.

[43] Douglas Thain, Todd Tannenbaum, and Miron Livny. Distributed computing in practice: the condor experience: Research articles. *Concurr. Comput. : Pract. Exper.*, 17(2-4):323–356, 2005.

[44] Yasuhiro Tsujimura and Mitsuo Gen. Genetic algorithms for solving multiprocessor scheduling problems. In *SEAL*, pages 106–115, 1996.

[45] Lee Wang, Howard Jay Siegel, Vwani P. Roychowdhury, and Anthony A. Maciejewski. Task matching and scheduling in heterogenous computing environments using a genetic-algorithm-based approach. *J. Parallel Distrib. Comput.*, 47(1):8–22, 1997.

[46] Ling-Ling Wang and Wen-Hsiang Tsai. Optimal assignment of task modules with precedence for distributed processing by graph matching and state-space search. *BIT*, 28(1):54–68, 1988.

[47] Annie S. Wu, Han Yu, Shiyuan Jin, Kuo chi Lin, and Guy Schiavone. An incremental genetic algorithm approach to multiprocessor scheduling. *IEEE Transactions on Parallel and Distributed Systems*, 15:824–834, 2004.

[48] E. Yildirim, I. H. Suslu, and T. Kosar. Which network measurement tool is right for you? a multidimensional comparison study. In *GRID '08: Proceedings of the 2008 9th IEEE/ACM International Conference on Grid Computing*, pages 266–275, Washington, DC, USA, 2008. IEEE Computer Society.

[49] E. Yildirim, D. Yin, and T. Kosar. Prediction of optimal parallelism level in wide area data transfers. *To appear in IEEE Transactions on Parallel and Distributed Systems*, 2010.

[50] Laurie Young, Stephen Mcgough, Steven Newhouse, and John Darlington. Scheduling architecture and algorithms within the iceni grid middleware. In *In UK e-Science All Hands Meeting*, pages 5–12, 2003.

[51] Albert Y. Zomaya, Chris Ward, and Benjamin S. Macey. Genetic scheduling for parallel processor systems: Comparative studies and performance issues. *IEEE Trans. Parallel Distrib. Syst.*, 10(8):795–812, 1999.

[52] Albert Y. Zomaya, Chris Ward, and Benjamin S. Macey. Genetic scheduling for parallel processor systems: Comparative studies and performance issues. *IEEE Trans. Parallel Distrib. Syst.*, 10(8):795–812, 1999.

Vita

Dengpan Yin was born in China. He transferred to the doctorial program in Louisiana State University, Baton Rouge after finishing his master's degree of Southern University in May 2008. He worked as a research assistant in Southern University from Spring 2007 to Spring 2008 and was honored as the Research Assistant of the Year in 2008. After that, he worked as a research assistant in the Center for Computation and Technology at Louisiana State University from August 2008 to December 2011. During the summer of 2010 he was employed as a software engineering intern by Futurewei Technologies, Santa Clara, California. His research interests include distributed systems, high performance computing and cloud computing. His publications related to the dissertation include:

- Yin, D., E. Yildirim, S. Kulasekaran, B. Ross and T. Kosar. 2011. A Data Throughput Prediction and Optimization Service for Widely Distributed Many-Task Computing. To appear in IEEE Transactions on Parallel and Distributed Systems- Special Issue on Many Task Computing

- Yin, D., and T. Kosar, 2011. A data-aware workflow scheduling algorithm for heterogeneous distributed systems. High Performance Computing and Simulation (HPCS), 2011

- Yildirim, E. , D. Yin and T. Kosar. 2010. Prediction of Optimal Parallelism Level in Wide Area Data Transfers . To appear in IEEE Transactions on Parallel and Distributed Systems.

- Yin, D., E. Yildirim and T. Kosar. 2009. A Data Throughput Prediction and Optimization Service for Widely Distributed Many-Task Computing. Proceedings of Many Task Computing in Grids and Supercomputers (MTAGS 2009) in conjunction with SC09, Portland, Oregon.

- Yildirim, E. , D. Yin and T. Kosar. 2009. Balancing TCP Buffer vs Parallel Streams in Application Level Throughput Optimization. Proceedings of International Workshop on Data-Aware Distributed Computing (DADC 2009) in conjunction with HPDC09, Munich, Germany.

- Kosar, T., M. Balman, I. Suslu,E. Yildirim, and D. Yin. 2009. Data-Aware Distributed Computing with Stork Data Scheduler. Proceedings of SEE-GRID'SCI'09, Istanbul, Turkey.