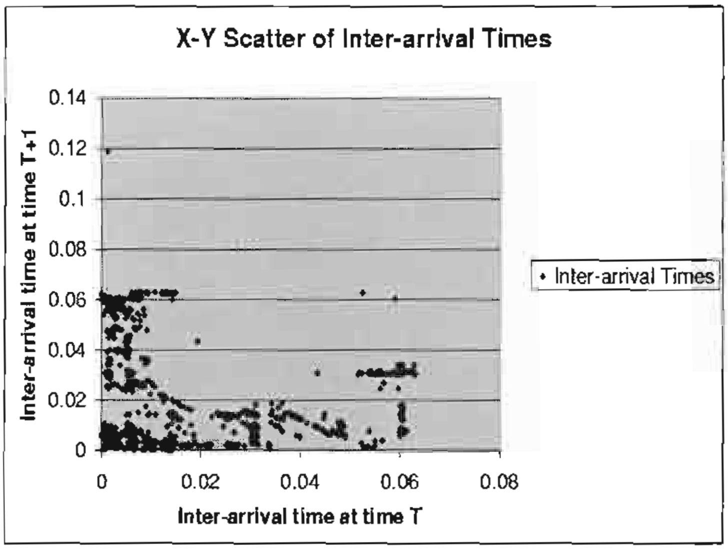

Figure 4: The X-Y scatterplot of one inter-arrival time with the next

sion time distribution. Since message transmission time is directly proportional to the length of the message, this parameter indirectly refers to the message length distribution. In an M/M/1 queueing system the service times are independent and exponentially distributed random variables. Thus, the following have to be true: the messages' lengths are exponentially distributed, the messages' lengths are independent from one another and the messages' lengths are independent of the time since the last arrival.

The message length varies from application to application depending on the characteristics of the application. We instrumented our test application, feature extraction and plotted the message length in a histogram, depicted in Figure 5. We determined that the distribution of message lengths is not exponential. However, it maybe independent of the particular application. Consequently, the distribution of the message lengths maybe independent of the workload in the system.

The data analysis presented in this section has been a case study for a single application and thus does not guarantee that the conclusions hold for other applications. The data analysis presented here leads to the conclusion that theoretically we are not able to apply an M/M/1 queueing theory model to our communication complexity models for prediction of overall performance of the parallel-pipeline model of execution. However, in the empirical results presented in this report, we show that in the context of the actual target application, grid scheduling, our complexity models predict execution times within a 10% average error interval of the actual execution times and a 15% optimal error interval of the actual execution times. Therefore, although not theoretically supportable, empirically it is reasonable to employ formulas inspired by queueing theory in our communication complexity models.

The analysis also show that we can apply the M/G/1 queueing theory model to the communication complexity model for the prediction. The predicted execution times from the model yeild the same figure of an average error interval of the actual execution times as in the M/M/1 model.

6 Application and Implementation

Our target application is feature extraction. Feature extraction is an important task in mining distributed textual data. We implement our parallel-pipeline model of execution using the feature extraction algorithms in HDDI [22], Hierarchical Distributed Dynamic Indexing, as our application. Our results confirm that the performance of the parallel-pipeline model of execution achieves a near-linear speedup on a dedicated homogeneous cluster.

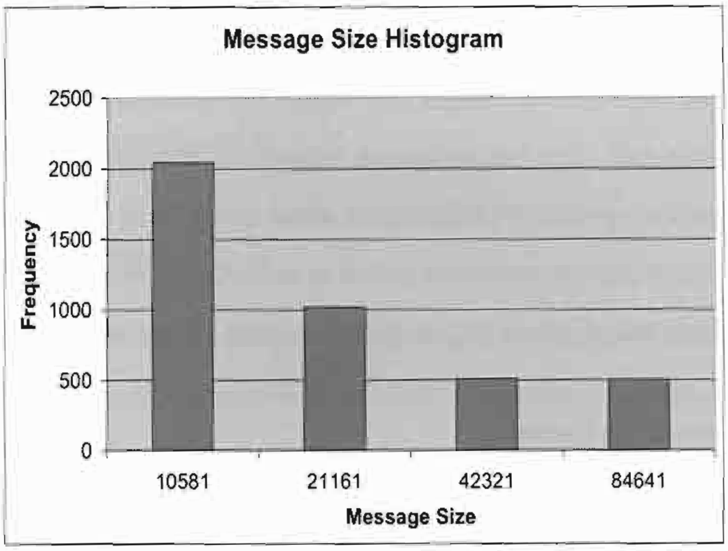

Figure 5: Histogram of message lengths in the parallel-pipeline model

6.1 Feature Extraction in HDDI

In this sub-section, we review the three functional parts of the HDDI feature extraction process: input, part-of-speech tagging, and concept extraction.

6.1.1 Input

Since a collection can originate from any source, we need to handle different input formats including SGML and various subsets such as HTML and XML. In addition, the feature extraction process requires us to identify particular fields of data in the input collection that are of interest (e.g., the title of an item). In order to accomplish these tasks we employed an extensible, reusable object-oriented input parser. See [3] for details.

6.1.2 Part-of-Speech Tagging

After identifying fields of interest, our feature extraction algorithms perform part-of-speech tagging. The part-of-speech tagger is a rule-based system for tagging English parts of speech. This system is based on [4, 5, 6]. The tagger uses three levels of rule sets to determine the part of speech of each word, and tags words with their English part-of-speech tag, as specified in the Brown tagset [10].

6.1.3 Feature Extraction

A key part of textual data mining is feature or concept extraction. For this purpose, we employed a sophisticated English language noun phrase extractor. Our premise is that maximal length noun phrases are high quality discriminators and should therefore be used as keyword features for indexing purposes by the HDDI textual data mining system. In order to identify maximal length noun phrases from the tagged text, a finite state machine capable of handling complex noun phrases was employed [3].

Concurrently with the extraction of noun phrases, other information that is used later in the HDDI model building stage is extracted and preserved. For example, a frequency of occurrence is calculated for each concept in each item as well as the character offset of each concept in the original item. Also, the field in which the concept occurred (e.g., title) is preserved.

Overall, the computation forms a global dictionary of noun phrase features. This is a reduction operation in three senses: first, the various sets of features are combined lexicographically in the merge stage. Simultaneously, occurrence frequencies for identical phrases (possibly from multiple documents) are reduced to a single frequency. Likewise, offsets for phrases occurring in multiple documents are merged. This creation of a global lexicon with occurrence frequencies and offsets is an associative reduction operation [21].

6.2 Implementation

In this sub-section we outline pseudo-code for the core computation and communication pattern of the implementation of our parallel, pipelined reduction model. The *while* loop in the code in Figure 6 implements continuous, never-ending execution of the core task as discussed in Section 3.2 for feature extraction. The *for* loop and the parameter *blksize* control the communication pattern described and depicted in Section 3 (Figure 1). The *if* clause determines whether a processor sends or receives a message. It is these loops that are (software) pipelined and executed in parallel.

```
MPI_Init(&argc, &argv);
MPI_Comm_size(&size);
P=size;
MPI_Comm_rank(&rank);
output=NP_extractor(url); or output=Sort(array); or output=Match(query);
while(true)
{
    blksize=2;
    for(i=1;i ≤ lg(P);i++)
    {
       if(rank%blksize>0 and rank%blksize≤blksize/2)
       {
          buf=output;
          dest=rank+blksize/2;
          MPI_Send(buf,dest);
          output=NP_extractor(url););
       }
       else
       {
          source=rank-blksize/2;
          MPI_Recv(buf,source);
          list=buf;
          Merge(output,list);
       }
       blksize=blksize*2;
    }
}
```

Figure 6: Pseudo-code for the Parallel-Pipeline Model implementation

7 Results on the IA-32 Cluster at NCSA

In this section, we detail the results of the application of our complexity model in a parallel-pipeline on a homogeneous computational cluster. As noted, one of our target applications is feature extraction from textual documents, an important task in mining distributed textual data. We employed our parallel-pipeline model of execution using the feature extraction algorithms implemented in HDDI [22].

7.1 Experimental Platform

The NCSA IA-32 cluster [24] is a cluster of 484 individual computing nodes, each with two CPUs per node, for a total of 968 processors. A high-speed, low latency Myrinet network interconnects the 484 compute nodes, and a Fast Ethernet network connects the cluster to file servers and the Internet.

7.2 Application of the Complexity Model

In this sub-section we detail the results of the application of our complexity model in a parallel-pipeline on a homogeneous computational cluster for our first target application, feature extraction from textual documents. As noted, this involves the creation of a global dictionary of lexicographically ordered features. Our results confirm that our performance prediction model is capable of estimating the resource requirements of this application when executed within our parallel-pipeline model of execution.

As discussed in Section 3, in order to maximize performance within the parallel-pipeline model of execution, the stages in the pipeline must be nearly equal and bounded by T_{Comp}. The number of processors participating in the parallel-pipeline is one of the parameters of the parallel-pipeline model of execution. The depth of the pipeline, for example, can be controlled by varying the number of processors in the pipeline in order to achieve maximum performance of the model. In addition, as discussed previously, the number of processors in the parallel-pipeline can be varied to control the value of T_{Merge}[4]. For this particular test application (feature extraction), we determined that executing the parallel-pipeline using 16 processors results in all stages being bounded above by T_{Comp}, and as a result the model achieves maximum performance.

The estimation of the execution time T_{Total} of our application within our parallel-pipeline model is calculated using Equations 6 and ?? presented in Section 4. Thus, the computation time T_{Comp} in our test application is the time required to extract features from a single textual input document and produce a sorted list of features. During the initial step (step 0) depicted in Figure 1, every processor executes this feature extraction task. Following this, starting in step one in Figure 1, $\frac{P}{2}$ processors continuously perform feature extraction and $\frac{P}{2}$ processors perform merging. In the experiments reported herein, the average input document size was approximately 5KB and the average computation time T_{Comp} for feature extraction was measured empirically to be 0.15 seconds. As noted, for this particular application $P = 16$ processors in the parallel-pipeline yields values of T_{Merge} and T_{Comm} that are bounded above by T_{Comp}.

Based on the use of 16 processors in the parallel-pipeline, we have determined L, the average message size, to be approximately $23,720$ bytes in this case. This yields an average for the actual communication complexity. Based on the Myrinet interconnection network in the IA-32, the channel capacity C is equal to 1.28 Gbits/second.

To understand the application of the prediction model, we now discuss an example. In one of the experiments that we conducted we employed 128 processors in

[4]This holds when the size of the data being merged grows with the depth of the pipeline (e.g., when merging multiple single-dimensional arrays during a parallel-pipelined sort).

the parallel-pipeline model. These processors were divided into eight groups of 16 processors each. Assuming a total of 4096 input documents, $\frac{4096}{8}$ documents were distributed to each set of 16 processors. As a result, the total time to process 4096 documents on 128 processors estimated by applying M/M/1 is[5]:

$$\begin{aligned} T_{Total} &= (\frac{N-P}{\frac{P}{2}} + 1) \cdot T_{Comp} + \frac{N-P}{\frac{P}{2}} \cdot T_{Comm} \\ &= (\frac{\frac{4096}{8} - 16}{\frac{16}{2}} + 1) \cdot 0.15 + \\ &\frac{\frac{4096}{8} - 16}{\frac{16}{2}} \cdot [\frac{\frac{23720*8}{1.28G}}{1 - (16 \cdot \frac{1}{0.15} \cdot \frac{23720*8}{1.28G})}] \\ &= 9.45 + 0.16 = 9.61 \ seconds \end{aligned} \tag{12}$$

Using the same example, we estimated the total time to process 4096 documents on 128 processors by applying M/G/1 as

$$\begin{aligned} T_{Total} &= (\frac{N-P}{\frac{P}{2}} + 1) \cdot T_{Comp} + \frac{N-P}{\frac{P}{2}} \cdot T_{Comm} \\ &= (\frac{\frac{4096}{8} - 16}{\frac{16}{2}} + 1) \cdot 0.15 + \\ &\frac{\frac{4096}{8} - 16}{\frac{16}{2}} \cdot [\frac{23720*8}{1.28G} + \frac{16 \cdot \frac{1}{0.15} \cdot (\frac{23720*8}{1.28G})^2}{2 \cdot (1 - 16 \cdot \frac{1}{0.15} \cdot \frac{23720*8}{1.28G})}] \\ &= 9.45 + 0.25 = 9.7 \ seconds \end{aligned} \tag{13}$$

It is clear that the communication time in this example (0.16 and 20 seconds) is of little engineering significance. Thus, our experiments with the feature extraction application serve mainly to test our predictions of T_{Comm} and T_{Total}.

We first employed 16 nodes on the IA-32 cluster. The results are presented in Table 1.

[5]For simplicity, we ignore the two lgP terms in Equation 6 that involve pipeline drain time.

Input Size	Parallel Execution Time	M/M/1 Predicted Time	M/G/1 Predicted Time
4096	83	77.97	77.98
8192	165	156.10	156.12
16384	326	312.36	312.41

Table 1: Results for 16 processors (in seconds)

In the second set of experiments, we used 32 nodes on the IA-32. These nodes were divided into two groups (communicators) of 16 processors each. The input was divided and distributed to each group equally. These groups of nodes concurrently executed our parallel-pipeline model. The results are presented in Table 2.

Input Size	Parallel Execution Time	M/M/1 Predicted Time	M/G/1 Predicted Time
4096	43	38.91	38.91
8192	83	77.97	77.98
16384	166	156.10	156.12

Table 2: Results for 16-processor sets using 32 processors (in seconds)

In the third set of experiments we employed 64 nodes. These nodes were divided into four groups of 16 processors each. As before, the input was divided and distributed evenly and the groups of nodes concurrently executed our parallel-pipeline model. The results are presented in Table 3.

Lastly, we employed 128 nodes on the IA-32 cluster. As noted previously, these nodes were divided into eight groups of 16 processors each. Again, the input was divided and distributed to each group evenly and the groups of nodes concurrently

Input Size	Parallel Execution Time	M/M/1 Predicted Time	M/G/1 Predicted Time
4096	23	19.37	19.38
8192	43	38.91	38.91
16384	84	77.97	77.98

Table 3: Results for 16-processor sets using 64 processors (in seconds)

executed our parallel-pipeline model. The results of the 128-node experiments are depicted in Table 4.

Input Size	Parallel Execution Time	M/M/1 Predicted Time	M/G/1 Predicted Time
4096	13	9.61	9.61
8192	23	19.37	19.38
16384	44	38.91	38.91

Table 4: Results for 16-processor sets using 128 processors (in seconds)

Table 5 summarizes the relative errors from the predicted time vs. actual execution time. The analysis reveals that the average absolute relative error is 11.5%. The analysis also reveals that this model does not produce an overestimation.

8 Conclusion and Future Work

In this research we have developed a framework that combines the speedup achieved from both parallel and pipelined execution in one model and hence per our theoretical result, achieves a near-linear speedup for parallelized associative operations. In

Num Proc	Input Size	Abs Rel Error M/M/1(%)	Abs Rel Error M/G/1(%)
16	4096	6.0530	6.0407
16	8192	5.3896	5.3772
16	16384	4.1811	4.1686
32	4096	9.5107	9.4991
32	8192	6.0530	6.0407
32	16384	5.9595	5.9472
64	4096	15.7496	15.7387
64	8192	9.5107	9.4991
64	16384	7.1714	7.1593
128	4096	26.0677	26.0585
128	8192	15.7496	15.7387
128	16384	9.5107	9.4991

Table 5: Relative Error on IA-32

order to achieve the optimum performance offered by our parallel-pipeline model, pipeline stages must be approximately equal in length.

One of the outstanding problems that we dealt with is the input reduction problem - bringing the input data to the nodes involved in the parallel-pipeline computation. To address this issue we have developed a framework for 'just-in-time' retrieval of the data needed for computation. This in effect adds another stage to the parallel-pipeline, and can also be modeled as part of the computational stage. Following completion of this effort, we implemented the final step, the send-to-server pipeline stage, in end-to-end systems for feature extraction and query processing.

We have developed a performance model that predicts the communication complexity for applications executing under our parallel-pipeline model of execution on a homogeneous computational cluster. We demonstrated the accuracy of these resource estimation models for a variety of processing environments and applications. Such models can provide information to a scheduler for a homogeneous computational cluster.

Our two prediction models yield very similar result and performance. The data analysis showed that it is theoretically supportable to emply formulas inspired by the M/G/1 but this is not true for the M/M/1. Empirically it is reasonable to employ formulas inspired by the M/M/1 queueing theory in our communication complexity model. The M/M/1 provides easier formulas and still gives the very similar performance as the M/G/1.

The performance of our prediction model varied. For feature extraction on the IA-32 cluster, the best predictions yielded an average error of about 10%.

One might ask if errors in the range of 10% to 20% are acceptable. It very much depends on the domain. We will take two examples from the world of queuing theory as guideposts:

1. Predicting the duration of a phone call to a call center is difficult; the variance is often quite large relative to the mean, so one would be happy with a prediction that was within 50%.

2. Predicting the duration of a manufacturing step, such as milling, is easier; one would only be happy with a prediction that was within 5%.

In our domain of parallel-pipeline execution, no previous studies have been performed on predicting run times. Therefore, our results in the 10% to 20% range are by default the state of the art. Predictions of this accuracy place the problem of scheduling multiple jobs somewhere between the world of M/M/c queues, with highly variable durations, and the world of job-shop scheduling, with durations of low variability.

We plan to continue to tune our resource estimation models by applying them to predict the resource utilization in end-to-end systems for information retrieval and feature extraction. This may require us to develop more sophisticated models of communication complexity at both ends of the parallel-pipeline. In addition, we plan to scale our parallel-pipeline model of execution and our resource estimation models to additional applications that involve associative operations. Finally, we expect to modify the feature extraction application so that it can merge or split input documents as required to balance the pipeline stages.

9 Acknowledgements

This research has been supported by the Thailand Research Fund.

References

[1] V. Adve. *Analyzing the Behavior and Performance of Parallel Programs.* PhD thesis, Univ. of Wisconsin-Madison, December 1993.

[2] Albert Alexandrov, Mihai F. Ionescu, Klaus E. Schauser, and Chris Scheiman. LogGP: Incorporating Long Messages into the LogP Model for Parallel Computation. *Journal of Parallel and Distributed Computing*, 44(1), 1997.

[3] R.H. Bader, M.R. Callahan, D.A. Grim, J.T. Krause, N. Miller, and W.M. Pottenger. The Role of the HDDI™ Collection Builder in Hierarchical Distributed Dynamic Indexing. In *Proceedings of Textmine '01 Workshop, First SIAM International Conference on Data Mining*, April 2001.

[4] E. Brill. A Simple Rule-based Part of Speech Tagger. In *Proceedings of the Third Conference on Applied Natural Language Processing.* ACL, 1992.

[5] E. Brill. *A Corpus-based Approach to Language Learning*. PhD thesis, University of Pennsylvania, Department of Computer and Information Science, 1993.

[6] E. Brill. Some Advances in Rule-based Part of Speech Tagging. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, 1994.

[7] M. Clement and M. Quinn. Analytical Performance Prediction on Multicomputers. In *Supercomputing 93*, 1993.

[8] D.E. Culler, R.M. Karp, D.A. Patterson, A. Sahay, E.E. Santos, K.E. Schauser, R.Subramonian, and T.von Eicken. LogP: A Practical Model of Parallel Computation. *Commun. ACM*, 39(11), 1996.

[9] H.P. Flatt and K. Kennedy. Performance of Parallel Processors. *Parallel Computing*, 12(1), 1989.

[10] W.N. Francis and H. Kucera. Brown corpus manual. Department of Liguistics, Brown University, 1979 revision, http://www.hit.uib.no/icame/brown/bcm.html.

[11] P.B. Gibbons, Y. Matias, and V. Ramachandran. Can A Shared-Memory Model Serve as a Bridging Model for Parallel Computation? In *Proceedings of the 9th ACM Symp. on Parallel Algorithms and Architectures*, 1997.

[12] J.F. JaJa and K.W. Ryu. The Black Distributed Memory Model. *IEEE Trans. Parallel And Distributed Systems*, 7(8), 1996.

[13] Sang Cheol Kim and Sunggu Lee. Measurement and Prediction of Communication Delays in Myrinet Networks. *J. Parallel and Distributed Computing*, 61, 2001.

[14] L. Kleinrock. On the Modeling and Analysis of Computer Networks. In *Proceedings of IEEE*, volume 81(8), 1993.

[15] J. Kuntraruk and W.M. Pottenger. Massively Parallel Distributed Feature Extraction in Textual Data Mining Using HDDI™. In *Proceedings of the Tenth IEEE International Symposium on High Performance Distributed Computing*, August 2001.

[16] V. Mak. Predicting Performance of Parallel Computations. *IEEE Trans. Parallel Distributed Systems*, 1(3), July 1990.

[17] A. Menasce and L. Barroso. A Methodology for Performance Evaluation of Parallel Applications on Multiprocessors. *J. Parallel Distributed Computing*, 14(2), 1992.

[18] J. Mohan. *Performance of Parallel Programs: Models and Analyses.* PhD thesis, Carnegie Mellon Univ., July 1984.

[19] D.A. Patterson and J.L. Hennessy. *Computer Architecture A Quantitative Approach.* Morgan-Kaufmann, 1996.

[20] William M. Pottenger. The Role of Associativity and Commutativity in the Detection and Transformation of Loop-Level Parallelism. In *Proceedings of the 12th ACM International Conference on Supercomputing*, July 1998.

[21] W.M. Pottenger. *Theory, Techniques, and Experiments in Solving Recurrences in Computer Programs.* PhD thesis, University of Illinois at Urbana-Champaign, Department of Computer Science, May 1997.

[22] W.M. Pottenger, Y. Kim, and D.D. Meling. *Data Mining for Scientific and Engineering Applications*, chapter HDDI™: Hierarchical Distributed Dynamic Indexing. Kluwer Academic Publishers, 2001.

[23] J. Schopf. Structural Prediction Models for High-Performance Distributed Applications. In *Cluster Computing Conference 97*, 1997.

[24] IA-32 Linux Supercluster. http://www.ncsa.uiuc.edu/userinfo/resources/hardware/.

[25] L.G. Valiant. A Bridging Model for Parallel Computation. *Communication of the ACM*, 33(8), 1990.

ภาคผนวก

Manuscript prepared to submit to

The 16th IEEE International Symposium on High Performance Distributed Computing (HPDC-16)

Communication Delay for a Parallel-Pipeline Model of Execution

Jirada Kuntraruk
Department of Mathematics, Statistics and Computer Science
Faculty of Science, Ubon Ratchathani University
jak5@sci.ubu.ac.th

Abstract

As computational cluster become viable alternative platforms for solving large computational problems, the research community acknowledges that the cluster environment can be used effectively when adaptive resource management is employed. This requires the ability to estimate the resource requirements of applications before scheduling decisions are made. We proposed a resource estimation model for applications executed in the parallel-pipeline model of execution. We study the use of the M/G/1 and M/M/1 queueing theory when applies to the communication models on the parallel-pipeline model. We propose the communication model that estimates the amount of time used to transfer the data through the cluster network. The proposed models were used to predict the communication time in the parallel-pipeline model. We compared the predicted time to the measured time from the experiments. An analysis of the average error in the prediction vs. actual execution time reveals that the proposed communication models were accurate with in 20

1 Introduction

Due to the wide variety of applications that can be parallelized within the associativity framework [16, 17], we developed a parallel-pipeline model of execution for computational clusters. This parallel-pipeline model of execution provides an execution framework within which applications involving associative operations can, in a limit, achieve near-linear speedups on homogeneous computational clusters.

In fact, as available computing power increases because of faster processors and faster networking, the computational cluster is becoming a viable alternative platform for executing distributed jobs to solve computational problems. It is recognized that a cluster can be effectively shared when adaptive resource management is employed. This implies an ability to estimate the resource requirements of any given run before a scheduling decision is made.

This paper thus addresses the problem of developing a resource estimation model for applications executed within our parallel-pipeline model of execution on a dedicated homogeneous computational cluster. The goal of this paper is to develop a practical performance model that predicts the resource utilization for applications executing under our parallel-pipeline model on a homogeneous computational cluster. There are two important factors that dominate the execution time in a parallel processing: the computation time and the communication time. We focus our study on the modeling the communication time utilized by the application running on our parallel-pipeline exeuction platform.

2 Related Work

A number of research projects [4, 5, 1, 3, 14, 13, 12, 19] have contributed useful results to the performance prediction of parallel computation in dedicated homogeneous environments. These prediction models are categorized as "classical" performance evaluation. The models treat two main components: the computation and the communication time. The work in [5] and [19], for example, provides a very simple communication delay model. It is, however, insufficient for our purposes. The models LogP [4], LogGP [2], BDM [8], BSP [20], and QSM [6] include some terms for network-related delays; they focus on upper bounds, and assume upper bounds for network delay are available without considering in detail how to derive them. Since we aim to provide data to schedulers as to *expected* job duration, especially in the presence of other jobs (and thus traffic) on the cluster, we must employ more sophisticated network contention models than employed in these previous works.

The communication delay model presented in [9] includes a network contention factor. Kleinrock [10] also introduced a method for applying queuing theory to model network contention in communication delays. We have developed performance models for a parallel-pipeline model of execution. We also characterize network contention in our performance models. We base our communication delay models on [9] and [10].

3 Parallel-Pipeline Model of Execution

We have developed a parallel-pipeline model of execution that performs a parallel reduction over a large set of distributed processors [11]. Reduction in this sense means a combining operation - for example, merging two sorted arrays in a merge-sort. The ability to perform a reduction in parallel relies on the fact that the target application involves one or more associative operations and can, as a result, be parallelized [16]. Therefore, theoretically, any application that involves an associative operation (e.g., a reduction) can be executed under our model. Computationally, the application can be modeled as two different tasks: first, a computational task, and second, a parallel merge as discussed in [11]. Of these two tasks, the parallel merge forms the reduction stage of the computation. In a distributed environment, communication takes place during the reduction. This communication is represented by the arrows in an example reduction pictured in Figure 1. The complexity of each step is modeled as *cost = computation time + communication time.*

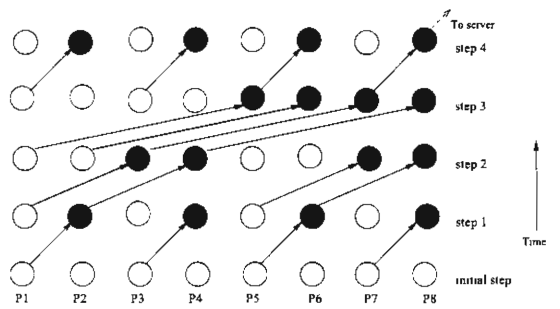

Figure 1. Parallel-Pipeline Reduction Model. The white nodes represent the execution of the application task and the black nodes represent the merging operation. This figure depicts execution on eight processors. The arrow edges represent the communication that takes place. The dotted lines and the arrow edges together form a reduction tree.

Figure 1 depicts an example of our parallel-pipeline model of execution on eight processors. During the initial step (step 0) every processor executes the application task. Then, starting with step one, a reduction is completed every $\lg P$ steps[1]. The system reaches a state of equilibrium after $\lg P$ steps. At each step afterwards, there are $\frac{P}{2}$ processors performing the application task and

[1]We make the simplifying assumption that the number of processors P is a power of 2. Note that $\lg P = \log_2 P$.

$\frac{P}{2}$ processors performing merges. There are $\lg P$ merge stages for each set of P processors in the parallel-pipeline because a complete binary tree with P leaves has a depth of $\lg P$. Since we model communication stages as part of the pipeline, the number of stages becomes $2 * \lg P$. Adding the initial stage pictured in Figure 1 yields, in this case, $2 * 3 + 1 = 7$ stages for a parallel-pipeline created from eight processors. This forms, in essence, a pipelined, parallel reduction consisting of $2 * \lg P + 1$ stages in which new input is continually being processed in the application task, and pipelined to the $2 * \lg P + 1$ stages of the reduction tree[2]. The lengths of the $2 * \lg P + 1$ stages in the pipeline are constrained such that all stages are equal, thus guaranteeing the optimality of the pipelined reduction [15].

3.1 Model Optimality

Due to the nature of the binary reduction tree, message size reaches a bound of $O(\frac{P}{2})$ when the computation reaches step $\lg P$ for many applications of interest. As noted previously, the system reaches a state of equilibrium that optimally uses the processors and communication resources given certain constraints on the target application. This optimal use of resources depends on the $2 * \lg P + 1$ stages being equal in length. These stages consist of T_{Comp}, $(\lg P - 1) * T_{Merge}$, $\lg P * T_{Comm}$ and $T_{CommServer}$ as depicted in Figure 2, where T_{Comp} is the time to perform the application task and T_{Merge} is the merge time for adding the result from a new task to the existing result.

As noted, the parallel-pipeline is optimal when all stages in the pipeline are equal in length and bounded above by T_{Comp}. To achieve this, for example, the number of processors participating in the merge operation in the parallel-pipeline can be used to control T_{Merge}. Similarly, T_{Comm} is dependent on message size, and for many applications T_{Merge} drives the size of messages (because greater fan-in during merge operations results in a larger output), and therefore T_{Merge} drives T_{Comm}. Consequently, the number of processors participating in a merge can be used to control T_{Comm} as well. Since T_{Comp}, T_{Merge} and T_{Comm} depend on the particular application, users can vary the number of processors participating in the parallel-pipeline in order to keep the pipeline stages balanced. A practical example of determining the number of processors needed to balance the pipeline stages is discussed in detail in Section 5.2.

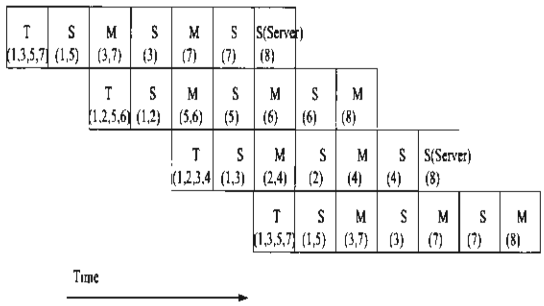

Figure 2. Parallel Pipeline. The parallel reduction pipeline of seven stages used in execution on eight processors. T is the application task executed on four processors. S is a send, M is a merge. Processors are numbered in parentheses in each stage of the pipeline.

4 A Performance Prediction Model for the Parallel-Pipeline Model of Execution

Although our original complexity model sketched in [11] is able to predict the behavior of our parallel-pipeline model of execution, it introduces unnecessary complexity. As a result, herein we develop a simplified complexity model, thereby making it easier to use the model for scheduling.

[2]Note that unlike a hardware pipeline, the communication between stages in the reduction tree is significant and as a result is modeled as $\lg P$ of the $2 * \lg P + 1$ stages.

Our new model is composed of the two components that play an important role in parallel program execution time, communication and computation. Figure 1 in Section 3 depicts the parallel-pipeline model on eight processors, where the white nodes represent the execution of the application task and the black nodes represent merge operations. During the initial step of execution in the parallel-pipeline, there are P tasks (i.e., input data items) processed on P processors, one task per processor. After this initial step, there are only $\frac{P}{2}$ tasks processed, since half of the processors are merging data received from the previous step. Therefore the number of steps required to process N input items (not including the initial step) is $\frac{N-P}{\frac{P}{2}}$. It takes lgP additional steps to drain the pipeline when using a binary reduction tree in the parallel-pipeline model. The computation that takes place in these last lgP steps is the merge operation. In this analysis, we ignore the last stage of the pipeline (the send-to-server). From Figure 1, it can be seen that almost every T_{Comp} or T_{Merge} stage has a matching communication stage (a send). The only exception is the final merge that takes place when the pipeline is drained. Therefore there is one less T_{Comm} stage than T_{Comp}/T_{Merge} stages in the parallel-pipeline.

Assume that we want to process N data input items (e.g., N single-dimensional arrays for sorting). The total time to process N input items is thus:

$$T_{Total} = (\frac{N-P}{\frac{P}{2}} + 1) \cdot T_{Comp} + lgP \cdot T_{Merge} + \frac{N-P}{\frac{P}{2}} \cdot T_{Comm} + lgP \cdot T_{Comm} \quad (1)$$

Per the optimality model presented in Section 3.1, T_{Merge} is bounded above by T_{Comp}. Therefore, we replace T_{Merge} with T_{Comp} in Equation 1. We do not, however, replace T_{Comm} with T_{Comp} even though the same optimality constraints hold. This is because T_{Comm}, as we will see, varies widely depending on the application, and the prediction model accuracy is improved by modeling T_{Comm} separately using queuing theory. This is the topic of the following section. Finally, as noted previously, the remaining lgP steps in Equation 1 comprise the pipeline drain time for both merge and communication stages.

4.1 A Delay Model using M/M/1 Queueing Theory

In the first communication compliexity model that we developed, M/M/1 queueing theory is applied to model the network contention in order to predict T_{Comm}. We will use the results of classical M/M/1 queueing theory to suggest functional forms for predicting communication delays. This classical queueing model assumes a Poisson stream of arriving messages requesting transmission over communication links, where each message has a length which is exponentially distributed with a mean of L bytes. The arrival stream in our applications will probably not be Poisson, but the M/M/1 formula may still give useful answers; thus, we will use it even though its assumptions may not be met. Let ρ denote the system utilization factor, then $\rho = \lambda \cdot \frac{L}{C}$ where λ is the arrival rate and C is the channel capacity. The mean response time of the system, as a function of L, is denoted $D(L)$ and is given by

$$D(L) = \frac{\frac{L}{C}}{1-\rho} + \tau \quad (2)$$

where τ is the propagation delay (i.e., the channel latency in seconds). In our situation, as in [10], the channel latency is negligible compared to $\frac{L}{C}$, so we set $\tau = 0$. The D(L) is therefore the cumminication complexity, T_{Comm}, for sending a message of size L over the network.

4.2 A Delay Model using M/G/1 Queueing Theory

In the second communication compliexity model that we developed, M/G/1 queueing theory is applied to model the network contention in order to predict T_{Comm}. We will use the results of classical M/G/1 queueing theory to suggest functional forms for predicting communication delays. This classical queueing model assumes a Poisson stream of arriving messages requesting transmission over communication links, where each message has a length which is exponentially distributed with a mean of L bytes. There is a difference apart from M/M/1. The M/G/1 model assumes that service times of a given application is dependent on the current workload in the system. Let ρ denote the system utilization factor, then $\rho = \lambda \cdot \frac{L}{C}$ where λ is the arrival rate and C is the channel capacity. The mean response time of the system, as a function of L, is denoted $D(L)$ and is given by

$$D(L) = \frac{L}{C} + \frac{\rho \cdot \frac{L}{C}}{2 \cdot (1 - \rho)} + \tau \qquad (3)$$

where τ is the propagation delay (i.e., the channel latency in seconds). In our situation, as in [10], the channel latency is negligible compared to $\frac{L}{C}$, so we set $\tau = 0$. The D(L) is therefore the cumminication complexity, T_{Comm}, for sending a message of size L over the network.

5 Results on the IA-32 Cluster at NCSA

In this section, we detail the results of the application of our complexity model in a parallel-pipeline on a homogeneous computational cluster. As noted, one of our target applications is feature extraction from textual documents, an important task in mining distributed textual data. We employed our parallel-pipeline model of execution using the feature extraction algorithms implemented in HDDI [18].

5.1 Experimental Platform

The NCSA IA-32 cluster [7] is a cluster of 484 individual computing nodes, each with two CPUs per node, for a total of 968 processors. A high-speed, low latency Myrinet network interconnects the 484 compute nodes, and a Fast Ethernet network connects the cluster to file servers and the Internet.

5.2 Application of the Complexity Model

In this sub-section we detail the results of the application of our complexity model in a parallel-pipeline on a homogeneous computational cluster for our first target application, feature extraction from textual documents. As noted, this involves the creation of a global dictionary of lexicographically ordered features. Our results confirm that our performance prediction model is capable of estimating the resource requirements of this application when executed within our parallel-pipeline model of execution.

As discussed in Section 3, in order to maximize performance within the parallel-pipeline model of execution, the stages in the pipeline must be nearly equal and bounded by T_{Comp}. The number of processors participating in the parallel-pipeline is one of the parameters of the parallel-pipeline model of execution. The depth of the pipeline, for example, can be controlled by varying the number of processors in the pipeline in order to achieve maximum performance of the model. In addition, as discussed previously, the number of processors in the parallel-pipeline can be varied to control the value of T_{Merge}[3]. For this particular test application (feature extraction), we determined that executing the parallel-pipeline using 16 processors results in all stages being bounded above by T_{Comp}, and as a result the model achieves maximum performance.

The estimation of the execution time T_{Total} of our application within our parallel-pipeline model is calculated using Equations 1 and ?? presented

[3]This holds when the size of the data being merged grows with the depth of the pipeline (e.g., when merging multiple single-dimensional arrays during a parallel-pipelined sort).

in Section 4. Thus, the computation time T_{Comp} in our test application is the time required to extract features from a single textual input document and produce a sorted list of features. During the initial step (step 0) depicted in Figure 1, every processor executes this feature extraction task. Following this, starting in step one in Figure 1, $\frac{P}{2}$ processors continuously perform feature extraction and $\frac{P}{2}$ processors perform merging. In the experiments reported herein, the average input document size was approximately 5KB and the average computation time T_{Comp} for feature extraction was measured empirically to be 0.15 seconds. As noted, for this particular application $P = 16$ processors in the parallel-pipeline yields values of T_{Merge} and T_{Comm} that are bounded above by T_{Comp}.

Based on the use of 16 processors in the parallel-pipeline, we have determined L, the average message size, to be approximately $23,720$ bytes in this case. This yields an average for the actual communication complexity. Based on the Myrinet interconnection network in the IA-32, the channel capacity C is equal to 1.28 Gbits/second.

To understand the application of the prediction model, we now discuss an example. In one of the experiments that we conducted we employed 128 processors in the parallel-pipeline model. These processors were divided into eight groups of 16 processors each. Assuming a total of 4096 input documents, $\frac{4096}{8}$ documents were distributed to each set of 16 processors. As a result, the total time to process 4096 documents on 128 processors estimated by applying M/M/1 is[4]:

$$\begin{aligned}T_{Total} &= (\frac{N-P}{\frac{P}{2}} + 1) \cdot T_{Comp} + \frac{N-P}{\frac{P}{2}} \cdot T_{Comm} \\ &= (\frac{\frac{4096}{8} - 16}{\frac{16}{2}} + 1) \cdot 0.15 + \\ &\frac{\frac{4096}{8} - 16}{\frac{16}{2}} \cdot [\frac{\frac{23720*8}{1.28G}}{1 - (16 \cdot \frac{1}{0.15} \cdot \frac{23720*8}{1.28G})}] \\ &= 9.45 + 0.16 = 9.61 \; seconds\end{aligned} \tag{4}$$

Using the same example, we estimated the total time to process 4096 documents on 128 processors by applying M/G/1 as

$$\begin{aligned}T_{Total} &= (\frac{N-P}{\frac{P}{2}} + 1) \cdot T_{Comp} + \frac{N-P}{\frac{P}{2}} \cdot T_{Comm} \\ &= (\frac{\frac{4096}{8} - 16}{\frac{16}{2}} + 1) \cdot 0.15 + \\ &\frac{\frac{4096}{8} - 16}{\frac{16}{2}} \cdot [\frac{23720 * 8}{1.28G} + \\ &\frac{16 \cdot \frac{1}{0.15} \cdot (\frac{23720*8}{1.28G})^2}{2 \cdot (1 - 16 \cdot \frac{1}{0.15} \cdot \frac{23720*8}{1.28G})}] \\ &= 9.45 + 0.25 = 9.7 \; seconds\end{aligned} \tag{5}$$

It is clear that the communication time in this example (0.16 and 20 seconds) is of little engineering significance. Thus, our experiments with the feature extraction application serve mainly to test our predictions of T_{Comm} and T_{Total}.

We first employed 16 nodes on the IA-32 cluster. The results are presented in Table 1.

InpSize	ParTime	M/M/1Time	M/G/1Time
4096	83	77.97	77.98
8192	165	156.10	156.12
16384	326	312.36	312.41

Table 1. Results for 16 processors (in seconds)

In the second set of experiments, we used 32 nodes on the IA-32. These nodes were divided

[4]For simplicity, we ignore the two lgP terms in Equation 1 that involve pipeline drain time.

into two groups (communicators) of 16 processors each. The input was divided and distributed to each group equally. These groups of nodes concurrently executed our parallel-pipeline model. The results are presented in Table 2.

InpSize	ParTime	M/M/1Time	M/G/1Time
4096	43	38.91	38.91
8192	83	77.97	77.98
16384	166	156.10	156.12

Table 2. Results for 16-processor sets using 32 processors (in seconds)

In the third set of experiments we employed 64 nodes. These nodes were divided into four groups of 16 processors each. As before, the input was divided and distributed evenly and the groups of nodes concurrently executed our parallel-pipeline model. The results are presented in Table 3.

InpSize	ParTime	M/M/1Time	M/G/1Time
4096	23	19.37	19.38
8192	43	38.91	38.91
16384	84	77.97	77.98

Table 3. Results for 16-processor sets using 64 processors (in seconds)

Lastly, we employed 128 nodes on the IA-32 cluster. As noted previously, these nodes were divided into eight groups of 16 processors each. Again, the input was divided and distributed to each group evenly and the groups of nodes concurrently executed our parallel-pipeline model. The results of the 128-node experiments are depicted in Table 4.

Table 5 summarizes the relative errors from the predicted time vs. actual execution time. The analysis reveals that the average absolute relative error is 11.5%. The analysis also reveals that this model does not produce an overestimation.

InpSize	ParTime	M/M/1Time	M/G/1Time
4096	13	9.61	9.61
8192	23	19.37	19.38
16384	44	38.91	38.91

Table 4. Results for 16-processor sets using 128 processors (in seconds)

NumProc	InpSize	M/M/1(%)	M/G/1(%)
16	4096	6.0530	6.0407
16	8192	5.3896	5.3772
16	16384	4.1811	4.1686
32	4096	9.5107	9.4991
32	8192	6.0530	6.0407
32	16384	5.9595	5.9472
64	4096	15.7496	15.7387
64	8192	9.5107	9.4991
64	16384	7.1714	7.1593
128	4096	26.0677	26.0585
128	8192	15.7496	15.7387
128	16384	9.5107	9.4991

Table 5. Relative Error on IA-32

6 Conclusion and Future Work

In this research we have developed a framework that combines the speedup achieved from both parallel and pipelined execution in one model and hence per our theoretical result, achieves a near-linear speedup for parallelized associative operations. In order to achieve the optimum performance offered by our parallel-pipeline model, pipeline stages must be approximately equal in length.

One of the outstanding problems that we dealt with is the input reduction problem - bringing the input data to the nodes involved in the parallel-pipeline computation. To address this issue we have developed a framework for 'just-in-time' retrieval of the data needed for computation. This in effect adds another stage to the parallel-pipeline, and can also be modeled as part of the computational stage. Following completion of this effort, we implemented the final step, the send-to-server pipeline stage, in end-to-end systems for feature extraction and query processing.

We have developed a performance model that predicts the communication complexity for applications executing under our parallel-pipeline model of execution on a homogeneous computational cluster. We demonstrated the accuracy of these resource estimation models for a variety of processing environments and applications. Such models can provide information to a scheduler for a homogeneous computational cluster.

Our two prediction models yield very similar result and performance. The data analysis showed that it is theoretically supportable to emply formulas inspired by the M/G/1 but this is not true for the M/M/1. Empirically it is reasonable to employ formulas inspired by the M/M/1 queueing theory in our communication complexity model. The M/M/1 provides easier formulas and still gives the very similar performance as the M/G/1.

The performance of our prediction model varied. For feature extraction on the IA-32 cluster, the best predictions yielded an average error of about 10%.

One might ask if errors in the range of 10% to 20% are acceptable. It very much depends on the domain. We will take two examples from the world of queuing theory as guideposts:

1. Predicting the duration of a phone call to a call center is difficult; the variance is often quite large relative to the mean, so one would be happy with a prediction that was within 50%.

2. Predicting the duration of a manufacturing step, such as milling, is easier; one would only be happy with a prediction that was within 5%.

In our domain of parallel-pipeline execution, no previous studies have been performed on predicting run times. Therefore, our results in the 10% to 20% range are by default the state of the art. Predictions of this accuracy place the problem of scheduling multiple jobs somewhere between the world of M/M/c queues, with highly variable durations, and the world of job-shop scheduling, with durations of low variability.

We plan to continue to tune our resource estimation models by applying them to predict the resource utilization in end-to-end systems for information retrieval and feature extraction. This may require us to develop more sophisticated models of communication complexity at both ends of the parallel-pipeline. In addition, we plan to scale our parallel-pipeline model of execution and our resource estimation models to additional applications that involve associative operations. Finally, we expect to modify the feature extraction application so that it can merge or split input documents as required to balance the pipeline stages.

7 Acknowledgements

This research has been supported by the Thailand Research Fund.

References

[1] V. Adve. *Analyzing the Behavior and Performance of Parallel Programs.* PhD thesis, Univ. of Wisconsin-Madison, December 1993.

[2] Albert Alexandrov, Mihai F. Ionescu, Klaus E. Schauser, and Chris Scheiman. LogGP: Incorporating Long Messages into the LogP Model for Parallel Computation. *Journal of Parallel and Distributed Computing*, 44(1), 1997.

[3] M. Clement and M. Quinn. Analytical Performance Prediction on Multicomputers. In *Supercomputing 93*, 1993.

[4] D.E. Culler, R.M. Karp, D.A. Patterson, A. Sahay, E.E. Santos, K.E. Schauser, R.Subramonian, and T.von Eicken. LogP: A Practical Model of Parallel Computation. *Commun. ACM*, 39(11), 1996.

[5] H.P. Flatt and K. Kennedy. Performance of Parallel Processors. *Parallel Computing*, 12(1), 1989.

[6] P.B. Gibbons, Y. Matias, and V. Ramachandran. Can A Shared-Memory Model Serve as a Bridging Model for Parallel Computation? In *Proceedings of the 9th ACM Symp. on Parallel Algorithms and Architectures*, 1997.

[7] IA-32Cluster. http://www.ncsa.uiuc.edu/userinfo/resources/hardware/.

[8] J.F. JaJa and K.W. Ryu. The Black Distributed Memory Model. *IEEE Trans. Parallel And Distributed Systems*, 7(8), 1996.

[9] Sang Cheol Kim and Sunggu Lee. Measurement and Prediction of Communication Delays in Myrinet Networks. *J. Parallel and Distributed Computing*, 61, 2001.

[10] L. Kleinrock. On the Modeling and Analysis of Computer Networks. In *Proceedings of IEEE*, volume 81(8), 1993.

[11] J. Kuntraruk and W.M. Pottenger. Massively Parallel Distributed Feature Extraction in Textual Data Mining Using HDDI™. In *Proceedings of the Tenth IEEE International Symposium on High Performance Distributed Computing*, August 2001.

12] V. Mak. Predicting Performance of Parallel Computations. *IEEE Trans. Parallel Distributed Systems*, 1(3), July 1990.

13] A. Menasce and L. Barroso. A Methodology for Performance Evaluation of Parallel Applications on Multiprocessors. *J. Parallel Distributed Computing*, 14(2), 1992.

14] J. Mohan. *Performance of Parallel Programs: Models and Analyses*. PhD thesis, Carnegie Mellon Univ., July 1984.

[15] D.A. Patterson and J.L. Hennessy. *Computer Architecture A Quantitative Approach*. Morgan-Kaufmann, 1996.

[16] William M. Pottenger. The Role of Associativity and Commutativity in the Detection and Transformation of Loop-Level Parallelism. In *Proceedings of the 12th ACM International Conference on Supercomputing*, July 1998.

[17] W.M. Pottenger. *Theory, Techniques, and Experiments in Solving Recurrences in Computer Programs*. PhD thesis, University of Illinois at Urbana-Champaign, Department of Computer Science, May 1997.

[18] W.M. Pottenger, Y. Kim, and D.D. Meling. *Data Mining for Scientific and Engineering Applications*, chapter HDDI™: Hierarchical Distributed Dynamic Indexing. Kluwer Academic Publishers, 2001.

[19] J. Schopf. Structural Prediction Models for High-Performance Distributed Applications. In *Cluster Computing Conference 97*, 1997.

[20] L.G. Valiant. A Bridging Model for Parallel Computation. *Communication of the ACM*, 33(8), 1990.