รายงานวิจัยฉบับสมบูรณ์

โครงการ: แบบจำลองความล่าช้าของการส่งข้อมูลของ Parallel-Pipeline Model ผ่านเครือข่าย โดยใช้แบบจำลองคิว M/G/1 และ M/M/1

โดย ดร.จิรดา กันทรารักษ์

ธันวาคม 2548

รายงานวิจัยฉบับสมบูรณ์

โครงการ: แบบจำลองความล่าช้าของการส่งข้อมูลของ Parallel-Pipeline Model ผ่านเครือข่าย โดยใช้แบบจำลองคิว M/G/1 และ M/M/1

ผู้วิจัย

ดร.จิรดา กันทรารักษ์
ภาควิชาคณิตศาสตร์ สถิติ และคอมพิวเตอร์
คณะวิทยาศาสตร์ มหาวิทยาลัยอุบลราชธานี

สนับสนุนโดยสำนักงานคณะกรรมการการอุดมศึกษา และ สำนักงานกองทุนสนับสนุนการวิจัย

(ความเห็นในรายงานนี้เป็นของผู้วิจัย สกอ. และ สกว. ไม่จำเป็นต้องเห็นด้วยเสมอไป)

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% error. The result shows that the performances of the M/G/1 and M/M/1 communication models were nearly identical.

1. ความสำคัญและที่มาของปัญหา

Pottenger developed a framework for understanding parallelism in a program based on the associativity of operations which accumulate, aggregate or coalesce a range of values of various types into a single conglomerate. The framework for understanding such parallelism is based on an approach that models loop bodies as coalescing loop operators. Within this framework we distinguish between associative coalescing loop operators and associative and commutative coalescing loop operators. We identified coalescing loop operators that are associative in nature in a variety of different applications. A number of these cases involve programs previously considered difficult or impossible to parallelize; however, the framework provides the necessary theoretical foundations for performing the analysis needed to prove these loops parallel and transform them into a parallel form.

Due to the wide variety of applications that can be parallelized within this theoretical framework, in follow-on work we developed a parallel-pipeline model of execution for computational cluster. This parallel-pipeline model of execution provides a parallel execution framework within which applications involving associative operations can, in the limit, achieve 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.

2. วัตถุประสงค์

This research 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 research 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 execution platform.

3. ระเบียบวิธีวิจัย

1. Study

ศึกษา M/G/1 Queueing Model

2.Apply

Apply M/G/1 Queueing Model กับ Communication Complexity ของ Parallel-Pipeline Model of Execution

3. Run Experiment

นำ Feature Extraction ซึ่งเป็น Application ตัวอย่างที่สามารถ Execute บน Parallel-Pipeline platform ได้มา Run เพื่อวัดเวลาในการ Run to completion

4. Validate

เปรียบเทียบ Complexity Model ที่ได้จากขั้นตอนที่ 2 กับผลการทดลองที่ได้จาก ขั้นตอนที่ 3 โดย ค่า Error ที่ได้ต้องอยู่ในเกณฑ์ที่น่าพอใจ

4. แผนการดำเนินงานตลอดโครงการ

กิจกรรม	*ระยะเวลา (เดือน)*	*ผลลัพธ์*
ศึกษา M/G/1 Queueing Model	2.5	ได้องค์ความรู้
Apply M/G/1 Queueing Model กับ Communication Complexity ของ Parallel-Pipeline Model of Execution	2	ได้ New Complexity Model ซึ่งเป็นทคโนโลยีใหม่
พัฒนาโปรแกรมเพื่อใช้ในการทดลอง	2.5	ได้โปรแกรมซึ่งเป็นเทคโนโลยีใหม่
ทำการทดลองเก็บข้อมูล runtime ของโปรแกรม	2	ได้ค่าตัวแปรที่จะใช้ใน Complexity Model
เปรียบเทียบ Complexity Model กับ	1.5	ได้ validate Complexity Model และค่า error

กิจกรรม	*ระยะเวลา (เดือน)*	*ผลลัพธ์*
ข้อมูล runtime จากผลการทดลอง		(ความคลาดเคลื่อน)
ขียนรายงานฉบับสมบูรณ์และบทความเพื่อตีพิมพ์ใน journal/ conference ระดับนานาชาติ	1.5	เานฉบับสมบูรณ์และบทความเพื่อตีพิมพ์ใน journal/ conference ระดับนานาชาติ

5. ผลงาน/หัวข้อเรื่องที่คาดว่าจะตีพิมพ์ในวารสารวิชาการระดับนานาชาติ

ชื่อเรื่องที่คาดว่าจะตีพิมพ์: Communication Delay Model for a Parallel-Pipeline Model of Execution

ชื่อวารสารที่คาดว่าจะตีพิมพ์: Proceedings of IEEE International Symposium on High Performance Distributed Computing

Modeling Communication Delay for a Parallel-Pipeline Model on Communication Network using M/G/1 and M/M/1 Queueing Model

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

1 Introduction

Due to the wide variety of applications that can be parallelized within the associativity framework [20, 21], 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 [8, 9, 1, 7, 18, 17, 16, 23] 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 [9] and [23], for example, provides a very simple communication delay model. It is, however, insufficient for our purposes. The models LogP [8], LogGP [2], BDM [12], BSP [25], and QSM [11] 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 [13] includes a network contention factor. Kleinrock [14] 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 [13] and [14].

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 [15]. 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 [20]. 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 [15]. 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 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

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

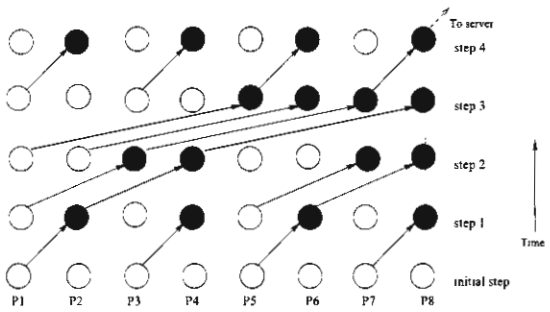

Figure 1: Parallel-Pipeline Reduction Model. The white nodes represent the execution of the application task and the black nodes represent the merging operation. This figure depicts execution on eight processors. The arrow edges represent the communication that takes place. The dotted lines and the arrow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

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 $\frac{P}{2}$ 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 [19].

[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.

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 7.2.

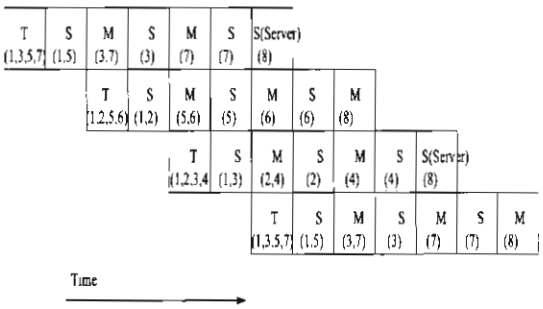

Figure 2: Parallel Pipeline. The parallel reduction pipeline of seven stages used in execution on eight processors. T is the application task executed on four processors. S is a send, M is a merge. Processors are numbered in parentheses in each stage of the pipeline.

3.2 The Speedup Model

In this sub-section we present the central theorem for the theoretical maximum performance of the parallel-pipeline model of execution.

Theorem 1: The parallel-pipeline model of execution achieves a near-linear speedup.

Proof: Let N be the number of tasks and P be the number of processors. Let T_{Comp} be the execution time for one task, T_{Merge} be an upper bound on the merge time, and T_{Comm} be an upper bound on the communication time for one or more tasks. We assume that $T_{Comp} \approx T_{Merge}$ during sequential execution and that $T_{Comp} \approx T_{Merge} \approx T_{Comm}$ during parallel execution (i.e., all pipeline stages are approximately equal[3]).

The speedup model incorporates speedups due to both parallel and pipelined execution as depicted in Figure 2 for an eight processor example. The sequential

[3]Note that the constraint of equal pipeline stages is required for optimality of the pipeline operation as discussed previously.

execution time is

$$T_{Seq} = N \cdot T_{Comp} + (N-1) \cdot T_{Merge} \tag{1}$$

The parallel execution time for one set of N tasks is

$$T_{Par} = \frac{N}{P/2} \cdot T_{Comp} + \frac{N}{P/2} \cdot (T_{Merge} + T_{Comm}) \cdot \lg P \tag{2}$$

The first term, $\frac{N}{\frac{P}{2}} \cdot T_{Comp}$, represents the execution time to produce results from N tasks using $\frac{P}{2}$ processors. The second term, $\frac{N}{\frac{P}{2}} \cdot (T_{Merge} + T_{Comm}) \cdot \lg P$, represents the reduction (combination) of the results from the $\frac{N}{\frac{P}{2}}$ sets of tasks. Each reduction of $\frac{P}{2}$ results in a set takes $\lg P - 1$ merges and $\lg P$ communications on a single set of $\frac{P}{2}$ processors, plus an additional merge or send-to-server, so the total reduction time for each set of $\frac{P}{2}$ tasks is $(T_{Merge} + T_{Comm}) \cdot \lg P$. Note that here we represent the final merge or send-to-server as a merge.

Generalizing from Figure 2 we have

$$Pipeline\ depth = 2 \cdot \lg P + 1 \tag{3}$$

This derives directly from the model. However, the actual maximum theoretical speedup is $2 \cdot \lg P$ due to a functional hazard in the first two stages of the pipeline. For example in Figure 2 processors 1,3,5,7 perform the application task in pipeline stage one, then 1,5 send results to 3,7, so none of these four processors are free until the end of stage two and no other processors are available because they are being used in other (e.g., reduction) operations when the pipeline is full.

The speedup due to parallel execution of one set of tasks on a single set of $\frac{P}{2}$ processors is

$$
\begin{aligned}
S_{Par} &= \frac{T_{Seq}}{T_{Par}} \\
&= \frac{P}{2 \cdot \lg P + 1}
\end{aligned}
\tag{4}
$$

The overall speedup is thus

$$
\begin{aligned}
S_{overall} &= S_{par} \cdot S_{Pipeline} \\
&= \frac{T_{Seq}}{T_{Par}} \cdot Pipeline\ depth \\
&= \frac{P}{2 \cdot \lg P + 1} \cdot 2 \cdot \lg P
\end{aligned}
\tag{5}
$$

Note that the $2 \cdot \lg P$ in the numerator is approximately equal to $2 \cdot \lg P + 1$ in the denominator. Thus, $S_{overall} \approx\leq P$, a near-linear speedup. This completes our proof.

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

Although our original complexity model sketched in [15] 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.

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:

$$T_{Total} = (\frac{N-P}{\frac{P}{2}} + 1) \cdot T_{Comp} + lgP \cdot T_{Merge} + \frac{N-P}{\frac{P}{2}} \cdot T_{Comm} + lgP \cdot T_{Comm} \quad (6)$$

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 6. 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 6 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

$$D(L) = \frac{\frac{L}{C}}{1-\rho} + \tau \tag{7}$$

where τ is the propagation delay (i.e., the channel latency in seconds). In our situation, as in [14], 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

$$D(L) = \frac{L}{C} + \frac{\rho \cdot \frac{L}{C}}{2 \cdot (1 - \rho)} + \tau \tag{8}$$

where τ is the propagation delay (i.e., the channel latency in seconds). In our situation, as in [14], 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 Queueing Theory for the Network Contention Model

The communication complexity models presented in this research report used the M/M/1 and M/G/1 queueing system. In this section we give an overview of the methodology and paremeters used to apply the queueing system to data communications networks. Finally, we present a case study to explore our assumptions in employing an M/M/1 queueing system.

5.1 Applying a Queueing System to a Network

There are a few key parameters in a data communications network system. The parameters that we are using are:

- C: capacity of the network
- L: message length

Let us assume that we have the classical queueing model of a Poisson stream of arriving messages requesting transmission over a communication link, where each message has a length which is exponentially distributed with a mean of L bytes. This implies that we assume a M/M/1 or M/G/1 queueing system. The system utilization rate factor or system service rate, ρ is thus:

$$\rho = \lambda \cdot (\frac{L}{C}) \tag{9}$$

where λ is the arrival rate at the network and C is the channel capacity. From the M/M/1 result, we know that the mean response time, T, of the system is calculated as:

$$T = \frac{\frac{L}{C}}{1 - \rho} \tag{10}$$

As for M/G/1, the mean response time, T, of the system calculated as:

$$T = \frac{L}{C} + \frac{\rho \cdot \frac{L}{C}}{2 \cdot (1 - \rho)} \tag{11}$$

Equations 9 and 10 are employed in our communication complexity models presented in Sections 4.1 and 4.2.

5.2 The Queueing Model and the Parallel-Pipeline Model

The M/M/1 and M/G/1 queueing system can only be applied to systems that meet certain criteria. Indeed, M/M/1 assumes a Poisson arrival rate and exponential service times. The M/G/1 also assumes a Poisson arrival rate but on the other hands assumes that the service times are independent of the workload in the systems. In this section, we perform a data analysis in order to determine if the parallel-pipeline model is a system that has Poisson arrival and exponential service times for a given application. This does not guarantee that the assumptions will hold for other applications, but rather provides one data point for discussion purposes.

5.2.1 The Arrival Rate and the Parallel-Pipeline Model

To determine if the parallel-pipeline model incorporates a Poisson arrival rate, we conducted a set of experiments that captured the times between successive arrival of messages in the data communication network, the inter-arrival times. For a Poisson process, these inter-arrival times are independent and are exponentially distributed random variables. In other words, the times between arrivals are exponentially distributed and the times between arrivals are independent.

The inter-arrival time that we are interested in here is the difference in arrival time of messages at the network. Therefore, the inter-arrival time is measured every time the send module is called in the application program.

To determine whether the inter-arrival times in the parallel-pipeline model are exponentially distributed, we plot the inter-arrival times in a histogram, depicted in Figure 3. In Figure 3, we see a graph that has an exponential flavor, but also has a lot more mass near zero. This leads to an inconclusive result. Therefore, we

tentatively conclude that for this particular application, the inter-arrival times might be exponentially distributed.

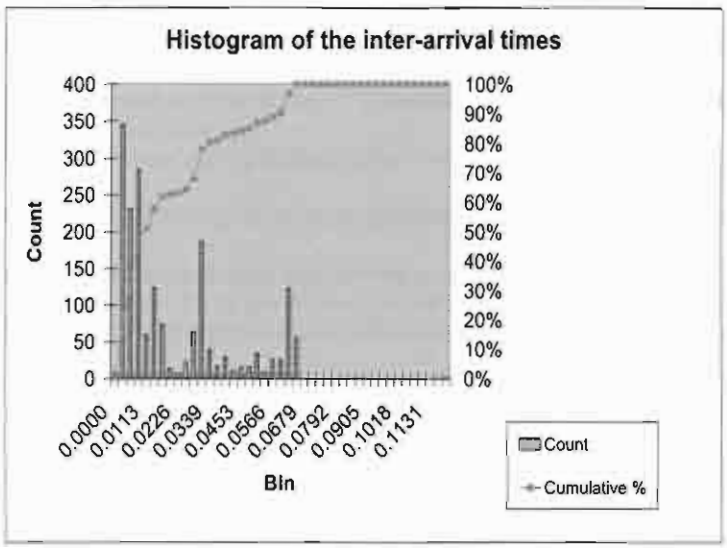

Figure 3: The histogram of the inter-arrival time in the parallel-pipeline model

To answer the question as to whether the inter-arrival times in the parallel-pipeline model are independent, we plot an X-Y scatter plot of inter-arrival times. The resulting graph in Figure 4 does not exhibit either a positive correlation (positive slope in the X-Y scatter plot) or a negative correlation (negative slope in the X-Y scatter plot). However, there are significant clustering effects. Thus, we conclude that the inter-arrival times in this particular application may be independent.

5.2.2 Service Times and the Parallel-Pipeline Model

Next we must determine whether the parallel-pipeline model of execution incorporates an exponential service time or an independent service time. As mentioned, the service time in a data communication network refers to the message transmis-