Realistic Communication Model for Parallel Computing on Cluster

Anthony T.C. Tam and Cho-Li Wang
System Research Group
Department of Computer Science and Information Systems
University of Hong Kong
{atctam, clwang} @csis.hku.hk

Abstract

In this paper, we present a model for parallel computation on commodity cluster environment. Our cluster model is targeted as a tool for both performance analysis and algorithm development. We abstract the communication event by means of local and remote data movements, and express all parameters by their associated cost functions instead of constants. Unlike previous models, we expose the contention problems explicitly and capture them in our parameters, thus enhance the ability to handle with contention issues.

We validate our model on a small commodity cluster by comparing the prediction accuracy of our model with the *Postal* model for the popular *log N* broadcast algorithm. We show that our model provides the answer to some performance issues that are missing in existing models, such as *LogP*, *Postal*, etc., and provides good prediction accuracy. For the design aspect, we also examine the gather collective operation, in which contention delay dominates its overall execution time. We have proposed a solution under our model for this problem that captures the congestion behavior with the ability to control the degree of contention and avoid the loss. Our results show that by scheduling the send processes according to the upper and lower bounds, the congestion behavior could be under our control and achieve optimal results.

1 INTRODUCTION

Using clusters as the alternate means of supercomputing is the current trend in high-performance computing. Its selling point is their cost-effectiveness as compared to traditional parallel machines. Besides, their performances are going along with the advances in commodity hardware, e.g. microprocessors and high-speed networks. Such a feature is an attractive point for parallel computing, but also be a problem in parallel algorithm design. Cluster computing is facing the same dilemma as traditional parallel computing – to design high-level, architectural independent algorithms that execute efficiently on a wide range of current and future cluster platforms. However, the rapid advances in hardware technology could hinder the portability and efficiency issues. Change of some hardware components may affect the performance ratio of the original design, which in turn may affect the efficiency of the programs. A simple example is the migration of the network component from Fast Ethernet to Gigabit Ethernet or even higher standards, the quantum leap in network performance could revoke many exist designs, as the network performance is catching up with the local data movement capability. The solution to the above challenge relies on the existence of a computation model that supports both performance analysis and algorithm design.

In this paper, we introduce a simple model for parallel computing on the cluster platform. The aim of our model is for performance analysis together with the ability to be an algorithm design tool, i.e. feasible for complexity analysis. Our model provides a rich set of performance parameters, which capture the crucial performance characteristics of the cluster, and the methodology to derive those parameters. Due to the dynamic nature of the cluster technology, change of cluster components may seriously affect the application run-time behavior. Therefore, we need to select appropriate parameters to analyze on the target architecture-application pairs. In our model, communication events are abstracted as some means of local and remote data movements, and each movement should have an associated cost and may be related to the length of the data items. Unlike previous models, we expose the contention problems explicitly and capture them in our parameters, thus enhance the ability to handle with contention issues.

Our works are organized as follows. In section 2, we briefly run through some of the existing models. Their strengths and weaknesses provide us with the correct direction in setting up our model. Section 3 describes the architectural background and assumptions of our model, together with a layout of all model parameters. In section 4, we validate our model on a small commodity cluster by comparing the prediction accuracy of our model with the LogP model for the popular *log N* broadcast algorithm. We show that our model provides the answer to some performance issues that are missing in those existing models, and provides good prediction accuracy. In section 5, we examine the gather collective operation, in which the contention situation dominates its performance. We propose a solution under our model for this problem that captures the congestion behavior together with the ability to control and utilize this phenomenon. Our results show that by scheduling the send processes according to the upper and lower bounds, the congestion behavior could under our control and achieve the optimal results. Finally, section 6 summarizes our work and concludes.

2 PARALLEL MODELS

In general, models have tended towards undesirable extremes. On one extreme, they are of highly theoretical qualities but be unrealistic or difficult to map onto real machines. At the other extreme, models may be too machine-oriented or complex which inhibit portability. The proposal of the *bridging model* [11] could not settle the issue easily, as the classification of being a bridging model is itself a debatable topic. Bear in mind that the definition of the model is *"simply an abstract view of a system or a part of a system, obtained by removing details in order to allow one to discover and work with the basic principles"* [4]. However, the complexity of designing and analyzing parallel systems requires that models be used at various levels of abstraction that are highly related to the application characteristics. Such that models should be developed for the relevant characteristics of the application together with the characteristics of the architecture. Hence, with such a diverge domain of applications designated for parallel computing, e.g. regular and irregular problems, a simple, rigid model could not serve for our needs.

The idea of setting up a model is for parallel algorithm designer to develop efficient algorithms for realizable, cost-effective architectures. If everyone have a common architecture insight, we can have a common ground for the development. Arguably or not, the architectural trend of parallel computers is converging toward the following scenario – Most modern large-scale machines are constructed from general-purpose nodes with a complete local memory hierarchy augmented by a communication assist, interfacing to a scalable network [3][8]. Existing models, including both abstract architecture models (e.g. LogP [3], BSP [11]) and communication models (e.g. Postal [1]), which targeting at the above architectural abstraction, have the following similarities:

- Introduced the importance of communication costs in parallel algorithm design.
- Assume reliable network, such that they treat sending a message as a *send-and-forget* [1] event.
- Assume fully connected network and the exact structure of the underlying communication network is ignored.
- Communication is based on point-to-point semantics, with the latency between any pair of processors roughly the same time for all cases.
- Performance characteristics of the communication networks are abstracted by a small set of parameters.

Under BSP Model, parallel algorithm is a sequence of parallel supersteps. Each superstep consists of a sequence of local computations plus any message exchange, followed by a global synchronization. For the LogP Model, it tends to be more network-oriented and simple. It uses four parameters to capture the cost associated with the communication events without limits to any programming style. An interesting feature of LogP model is the idea of finite capacity of the network, such that no more than certain amount of messages can be in transit from any processor or to any processor at any time. Any attempts to exceed the limit will stall the processor. However, the model does not provide any clear idea on how to quantify, avoid and take advantage of this information in algorithm design. The Postal Model is similar to LogP model with the exception of more abstractly express the network. The system is characterized by two parameters only, and this effectively reduces the dimension of analysis. Therefore, it facilitates communication analysis rather than for performance studies.

These models provide an abstract ground for development, however, they have the some drawbacks. First, in the realistic environment, sending a message is not as simple as send-and-forget event. Although including this assumption simplifies the analysis, but this could affect the overall prediction accuracy in performance analysis. Second, they assume simultaneously send and receive operations in unit time step. This assumption has significant impact on the performance of some communication operations, as in reality, interference exists between send and receive events. Third, their parameters neglect factors related to message size, communication load, and contention issue, which influence the communication latency in a large degree in real environment. Forth, too restricted set of performance parameters reduces their ability to make accurate prediction of real performance, and also restricts the elucidative power of the models. Fifth, restricted to certain programming style, such as the need to have global synchronization between supersteps, may affect their overall usage. Lastly, their cost models are better for asymptotic analysis than for prediction, when porting the resulting algorithm to a particular platform, significant efforts have to be made for tuning the algorithm for performance.

Most of the above drawbacks come from the tradeoff between simplicity and accuracy. To be more realistic, we can include more parameters to the model. Parameters are not always important for all situations; the used of those parameters are subjected to the target level of abstraction that we are going to work on. For instance, using a simple *latency* parameter may be good enough to capture the cost of the point-to-point communication, but is too simple for explaining the many-to-one or many-to-many issues. Therefore, under some circumstances, a few performance parameters are proved to be adequate for modeling the parallel system. And on other situations, there are other issues that need to be studied or included for making the correct judgement. As a result, it is sensible to think that models should provide some mechanisms to improve theirs expressiveness when needed.

3 OUR PERFORMANCE MODEL

We believe that the use of the model in algorithm analysis should be done straightforwardly and easily, whenever the users are provided with some systematic means of analysis. Emphasis has been made on the derivation of those parameters by software approach, which is the key to the whole analytical process. Based on these measurable parameters, high level primitives can be built or analyzed, and these primitives can also be used as some high-level performance parameters in analyzing complicated applications. Our approach is similar to the LogP model, which exposes crucial performance parameters; however, we opt to provide a richer set. Furthermore, instead of using constant values, we capture these parameters as some cost functions, in which, both the message length, traffic load, and contention factors are included if appropriate. Simple methods are included in our discussion for deriving those functions. We believe that with a larger set of measurable low-level parameters, more useful information can be obtained for efficiently tuning of the parallel applications.

3.1. Architectural Viewpoint of Cluster

In our model, a cluster is defined as a collection of autonomous machines that are interconnected by two separate networks, one is the traditional reliable local area network (LAN), and the other is the high-performance system area network (SAN). Parallel applications running on the cluster make use of both networks, the SAN is used merely as the communication network for the parallel applications, while the LAN is used as the control/backup network, e.g. job submission, cluster management, etc. For non-parallel applications such as sequential and distributed tasks, they will make use of the LAN system if communications between machines are needed. Thus our communication model is targeting to the performance characteristics of the SAN network under parallel applications.

To reduce the intractability, we have to limit the analytical space with the following assumptions. All the processors have the same local characteristics, such as computation power, memory hierarchy, operation system supports, and communication hardware – homogeneous clusters. In general, parallel programs are programmed in SPMD mode, and follow the compute-interact mode of coarse-grained synchronization. As for the communication (SAN) system, it involves a simple router switch, interconnecting all nodes, and assumes complete graph topology; hence, the network diameter between cluster nodes is not relevant to the performance studies. For the switch architecture, we assume it is a packet-switched, synchronous, pipeline network, with cut-through feature. The logical unit for communication is a packet, which is bound by the range [1..*MTU*[1]]. To send long messages, the system transfers them as sequences of packets. The switch element has constant delay in routing the packets from any input ports and output them in the absence of *conflicts*, but the overall performance is affected by the traffic load. Conflicts take place if more than one packet need to access the same output line, and are resolved by always forwarding the first arrival packet, and the rest are buffered and routed later on by the switch. The amount of buffer memory inside the switch is assumed to be finite, thus, the network can sustain certain level of congestion rather than just as $\lceil L/g \rceil$ packets in the case of the LogP model. At first instance, this global router concept [9] seems to be a stringent assumption as this limits the scalability aspect of the cluster systems. However, current network technology allows us to scale the switching system to support hundred of machines interconnected by a single router switch with more or less similar point-to-point latency [5].

3.2. Cost Model

For our cost model, we are focusing on the costs induced by moving data around, both locally and remotely. We consider data communication via the network as an extension to the concept of memory hierarchy, such as a movement from the remote memory region to the local memory region. So there are two types of data movements involved in the parallel computation.

Remote Data Transfer
We abstract the data movement via the global network by three phases, from sender address space to receiver address space. First, data on the sender side traverse through a *send phase*, which is initiated by the host processor. Then the network delivers the messages to the other end during the *transfer phase* of the communication. At remote end, *receive phase* consumes the data and terminates the transfer event. We encapsulate this event by the following parameters.

- *p* – this refers to the number of processors participate in the current event.
- *Send Phase* - O_s

This is a synchronous event as it is initiated by the sender. From the high level perspective, we can view it as the time used by the user process to interact with the logical network interface, prepare the message, queue it to the send queue, and signal the network hardware. The overall cost of this phase reflects the processing speed of the host CPU, the efficiency of the memory subsystem, and virtual interface protocol in use. We model this phase by a simple linear function, $O_s(m) = \kappa_s + \tau_s m$, where κ_s is the startup cost of this event and is depended on the host processing power, *m* is the message length bounded by the range [1..*MTU*], and τ_s is the data transfer rate that depends on the efficiency of the

[1] *MTU* stands for the Maximum Transfer Unit of the communication system.

memory subsystem. Subject to the communication protocol, e.g. no memory copy is involved, this linear function can be reduced to a simple constant, i.e. $O_s(m) = \kappa_s$. We quantify this event by *directly* measure the time engaged by the CPU in handling this activity.

♦ *Send Gap - g_s*

Owing to the difference in data movement speeds between the Send and Transfer phases, two consecutive packets cannot be served by the network hardware within certain time interval. This inter-packet *gap* has two meanings. First, it reflects the moving capability of the network with respect to the host processing speed. Thus, the difference between O_s and g_s indicates the amount of CPU cycles available for the processor to do other useful computation. Second, it represents the maximum per-processor inter-arrival rate of the packets to the router. This parameter is delineated as, $g_s(m) = g_1 + \tau_1 m$, where g_1 stands for the startup cost necessary to initiate the transfer and τ_1 reflects the available per processor communication bandwidth. Due to the limited buffers in the send queue, if a sender generates messages faster than the network can dispatch, new message can only be accepted if the network has just finished servicing one. By quantifying this servicing time with respect to the message size, the required cost function can be obtained.

♦ *Transfer Phase – L*

This parameter represents the time used by the network in delivering data from the source memory to the destination memory. For example, from the send queue of local machine to the receive queue of the remote machine. It is a network-dependent parameter, and it encapsulates the transmission speeds of different network components, the diameter between the communicating entities, the network topology and the network protocol in use. As we model the network as a complete graph with single global routing switch, both diameter and topology factors can be eliminated. In a realistic environment, its performance is subjected to the traffic loading at any particular instant. For example, when routing a packet through the switch, if the outgoing port is engaged, temporary buffering is needed. This delay affects the overall network performance perceived by the users. We model this phase by the following bilinear function under *congestion-free* condition, $L(m,p) = l(p) + \tau_l(m,p)m$, where $l(p)$ is a function represents the cumulative startup cost of this transfer and $\tau_l(m,p)$ is the observed network throughput of the peer-to-peer communication. Both l and τ_l are a function of p. Routing a packet involves utilization of some central resources (e.g. buffer control unit, arbitration unit), therefore, contention for resources may occur if more than one routing request happen concurrently. The extent of this contention is subjected to the switch internal architecture, and different routers may have different relations on p. Due to the limited aggregate bandwidth, the router cannot support many communicating pairs at a time and contention arises. So the perceived local bandwidth depends on the available aggregate bandwidth together with the volume of the communication, thus, τ_l is a function of both p and m. To measure it by software means, we have to calculate it indirectly. Based on the fact that a *pingpong* pair involves $2*(O_s+L+O_r+U_r)$ time units, if we know all other parameters, $L(m,1)$ can be calculated. By artificially generate multiple concurrent pairs of pingpong nodes, the corresponding bilinear function could be obtained by using linear regression.

♦ *Receive Gap – g_r*

This parameter stands for the *minimum* gap between two consecutive receptions experienced by the receiving host. In other words, it is the maximum inter-arrival rate of the packet delivery by the network. This parameter has two uses. First, this gap reflects the CPU cycles available to handle arrived packets, so protocol design or other computation can utilize this information. Second, this gap relates to the minimum service time of the router in delivering packets, as we cannot receive more than one packet in that interval. So inclusion of this parameter is crucial for performance analysis. Same as the g_s parameter, it is captured by this function, $g_r(m) = g_2 + \tau_2 m$. For simplicity, we can generally assume $\tau_1 = \tau_2$, as both are related to the transfer capability of the network router. To quantify this parameter, we can converge multiple streams of data to a single receiver, and measure the minimum time perceived by the receiver in detecting the arrival of data packets.

♦ *Network Buffer – B_L*

Limited resources are the major cause of congestion, which in turn, affect the delay experienced by the applications. In reality, congestion is a fact that we need to face with instead of neglecting it. This parameter reflects the available buffers in the global router. By capture the finite capacity of the network buffers, algorithm designer can calculate the network endurance, and avoid contention loss with appropriate communication schedule. To quantify this parameter, we perform a set of tests which floods the router under different duration, and record the percentage of packet arrival at the destination. Then, by mapping the data with Equation 4 (appeared in section 5), we can estimate the buffer capacity of the router switch.

♦ *Asynchronous Receive Phase – O_r*

This parameter captures the software overhead in handling incoming messages. This phase is handled by kernel thread and does not involve the receiving process, so it is considered to be an asynchronous event. Likely, the performance is affected by the processing speed of the processor, the algorithm used, the signaling mechanism, and the I/O bandwidth.

In our model, we express it as a linear equation, $O_r(m) = \kappa_r + \tau_r m$. In which κ_r represents the minimal cost of this asynchronous event, such as interrupt cost, buffer management, and protocol overhead; while τ_r reflects the memory movement between different memory regions if needed. As it is an asynchronous event, which triggers by a remote partner, we have difficulty in direct measuring its cost. Observed that during the reception, system resources are consumed, therefore, other workloads would be affected. Under experimental control, we measure the execution time of a control computation, and then we measure another runs of this computation segment with message reception happens in the background. We can then estimate the increase in execution time due to this message reception. Thence, we have indirectly measured the induced software overhead.

- *User Receive Phase – U_r*

Due to the asynchronous nature of the communication, the receiving process needs to find some means to check for data arrival, e.g. polling, block & wake-up by signal, or hybrid approaches; and consumes the data, e.g. copy to other memory segment. This parameter reflects the cost spent by the receiving process after arrival of messages. In most of the performance evaluation reports, due to the artificial nature of the benchmark programs, this phase is of insignificantly low cost. However, in real parallel computation, this phase can introduce large delay to the communication event if not coordinated properly. For example, in a non-dedicated cluster environment, polling is a user-level event that is affected by the regular CPU scheduling policy. If the receiving process cannot be scheduled frequently to poll for its data, the overall performance may degrade a lot.

Local Data Transfer

- *Memory Copy overheads – M_{ctc}, M_{ctm}, & M_{mtm}*

Memory copy issue has been extensively studied in the past, and is being classified as high overhead event. To avoid this overhead, most of the high-performance communication systems have removed it from their protocol stacks. However, in reality, memory copy operations cannot be avoided completely. To quantify these costs, we provide three memory copy parameters - M_{ctc}, M_{ctm}, and M_{mtm} to represent the costs induced by data movement between different memory hierarchies, such as the cache-to-cache, cache-to-memory, and memory-to-memory data movement.

4. VERIFICATION OF THE MODEL

4.1. Experimental Platform

Our experimental platform is a cluster consists of 16 standard low-end PCs running unmodified version of the Linux 2.0.36 operating system. Each node contains a 200MHz IDT Win-Chip processor with 256KB external L2 cache and 32MB of main memory. Our interconnecting network is the commodity Fast Ethernet components driven by our own DP communication system [7]. Each node includes a DEC21140-based Fast Ethernet card and is connected by a 24-ports Intel 510T stackable Fast Ethernet switch. The model parameters for our cluster system are shown in Table 1. These formulae capture the major performance characteristics of our experimental platform on both hardware and software aspects. All parameters are derived by means of regression analysis on the data collected from our benchmark tests described in previous section.

Parameters	Cost Formulae (μs)	
	$m \le 40$	$m > 40$
O_s	0.0291 * m + 6.308	0.0253 * m + 6.37
T_s	0.049 * m + 11.732	0.0776 * m + 10.88
g_s	6.73	0.0796 * m + 3.027
g_r	6.73	0.0788 * m + 4.986
O_r	0.0426 * m + 12.245	0.0429 * m + 12.176
U_r	9.93	
L	$1.556 * p + 16.684 + 0.08264 * max\left(1, \frac{p * m}{90 * g_s(m)}\right) * m$	
mc_{ctc}	0.015 * m + 0.194	
mc_{ctm}	0.0149 * m + 2.07	
mc_{mtm}	0.0314 * m + 1.756	
B_L	1935 (units)	
BW_{agg}	800 Mbit/s	

Table 1. Cost formulae of the measured parameters

4.2. Broadcast Operation

Broadcast operation is a one-to-many communication event, which is a common event happens in parallel and distributed applications. If the underlying hardware does not support broadcast operation, it has to be implemented by the point-to-point primitive. Efficient broadcast algorithms on various architectural models have been developed, such as in [1][6][10], and they are claimed to be (sub)optimal under their parallel models. But in reality, when they are implemented in real platforms, some performance may be lost. There are two reasons. First, the proposed algorithms may be mathematically achievable, but difficult to be implemented correctly and efficiently in real world. Second, some characteristics of the real systems are being simplified, which result in unexpectedly increased in overheads and latencies. These minor perturbations may only be surfacing in real situations, but not appear in their simplified world. Nevertheless, those optimal algorithms are seldom being used in traditional parallel programming packages such as MPICH, LAM, CHIMP, etc. Most of these communication libraries are using simple algorithms such as linear algorithm or *log N* algorithm. To improve the qualities of these communication libraries, either users have to implement these *optimal* algorithms by their own, or they perform tuning on those simple algorithms for the best of their working platforms.

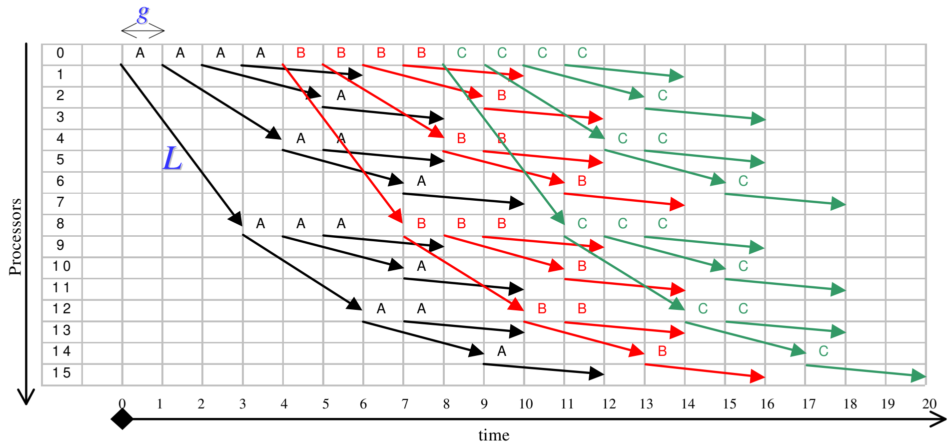

Figure 1 Broadcast tree constructed on the LogP model for p=16, L=3, g_s=1, O_s=0

One of the attractive points of including more parameters in the model is that we now have a more powerful *meter* to analysis algorithms on the target architecture, especially on performance analysis. We have implemented a broadcast operation on our experimental platform, which is the basic *log N* algorithm. The major reason to examine this algorithm is its simplicity and popularity. To broadcast a *long message* (larger than the MTU), we simply *repeat* the *log N* algorithm *k* times, each carry a portion of the message. Based on the LogP model, we have constructed the communication tree and is shown in figure 1, and the corresponding *Cost* formula is:

$$B_k(P) = (k-1)\cdot log_2 P \cdot g_s + B_1(P) = (k-1)\cdot log_2 P \cdot g_s + log_2 P \cdot L = log_2 P \cdot (kg_s + L - g_s) \qquad (1)$$

From the above cost formula, the running time of this algorithm grows linearly with *k*, which is not optimal. As we can clearly discover a lot of empty time-slots from the communication schedule, and this explains the non-optimality of this algorithm. Thus, the theoretical optimal solution to this multiple broadcast is to fill up those empty time-slots as much as possible to minimize the overall cost. One of these optimal algorithms is reported in [6]. Even though the *log N* algorithm is not optimal, we should expect to have precise performance prediction returned by LogP model. To verify this, we have collected the timing results of this algorithm on our experimental platform for different number of nodes and message lengths. The results are shown in figure 2 with both measured and predicted times are displayed.

The results showed that for small size messages, the LogP model could closely predict the performance. However,

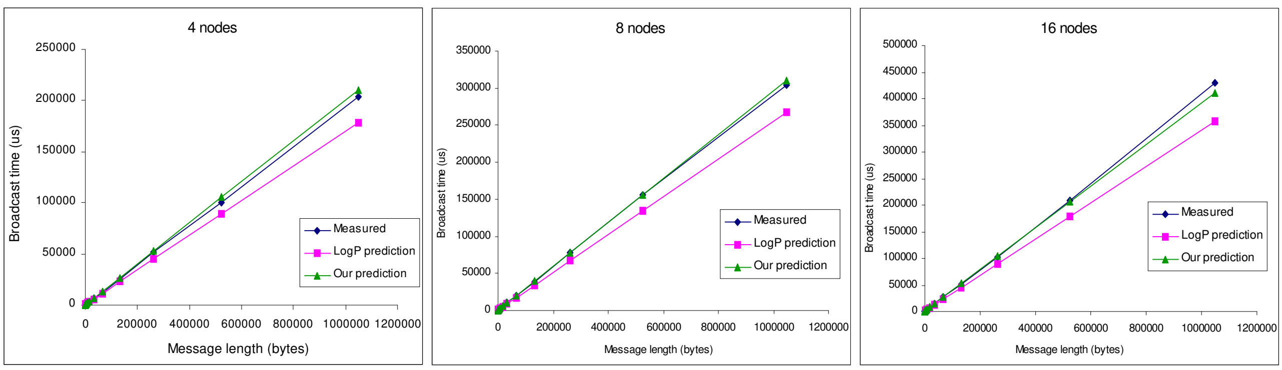

Figure 2 Measured and estimated execution time of the broadcast operation on our cluster. This figure compares the prediction accuracy between our cluster model with the LogP model.

when the message length increases together with the increase in the number of participating nodes, the prediction error raises remarkably. From the cost formula, it cannot explain what is the cause of the deviation, as it only has limited number of model parameters. To explain those deviations, let's take a look at the communication schedule of this broadcast algorithm that based on our cluster model (figure 3).

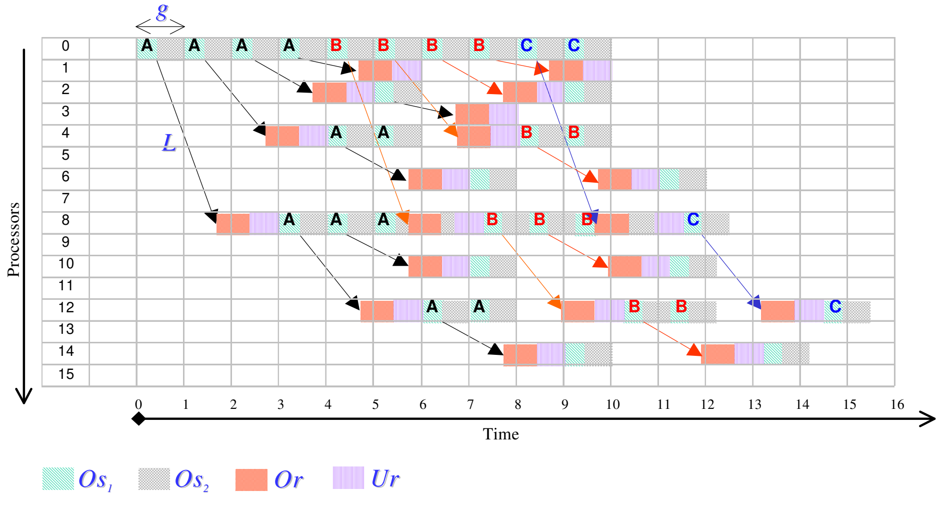

Figure 3 The communication schedule of the broadcast operation on our cluster model with O_s=0.4, L=1.3, O_r=0.6 and U_r=0.7.

From our model, we identify the bottleneck region of this broadcast pattern is at the P/2th processor, but not at the broadcast root processor. Of the previous broadcast communication schedule (figure 1), we find no interference exists in successive broadcast, such that for those internal nodes, the reception of the *i*th message does not overlap with the transmission of the *i-1*th message to its subtree. But in reality, as shown in figure 3 at processor 8 – the *P*/2th processor of this broadcast event, the transmission of the ith message is always overlapping with the reception of the *j*th message for any $j>i$. This observation comes from the fact that the assumption of simultaneous send and receive operations have been violated. From the low-level perspective, both transmission and reception involve data movements between system memory and the network. Within the data movement path, lots of resources are shared, hence, contention arises if a node is actively sending and receiving messages. When the number of successive broadcast (*k*) is small, this interference may be of negligible value, so the normalized LogP/postal model could predict the behavior accurately. When *k* becomes larger and the user process cannot handle those arrived messages immediately, as a result, the delay becomes larger and this is being reflected by the large predicted error depicted in figure 2.

With the above observation, the new Cost formula of this broadcast pattern that based on our cluster model becomes,

$$\begin{aligned} B_k(P) &= Os_1 + L + Or + Ur + (k-1)\cdot\left[(\log_2 P/2)\cdot g + Or + Ur\right] + B_1(P/2) \\ &= Os_1 + L + Or + Ur + (k-1)\cdot\left[(\log_2 P/2)\cdot g + Or + Ur\right] + (Os_1 + L + Or + Ur)\cdot \log_2 P/2 \\ &= \log_2 P\cdot(Os_1 + L + Or + Ur) + (k-1)\cdot\left[(\log_2 P/2)\cdot g + Or + Ur\right] \end{aligned} \tag{2}$$

When comparing both cost formulae, our cost model uncovers the overhead induced by the message reception of each successive broadcast. And we show that this hidden overhead is proportional to the number of broadcast (*k*). This explains why the prediction accuracy of the previous cost formula deteriorates when *k* becomes large. In figure 2, we have applied the corresponding model parameters to our cost formula, and shows that our prediction accuracy improves significantly.

5. APPLICATION OF THE MODEL

5.1. Gather Operation – Algorithm Design/Performance Tuning

Gather operation is a *many-to-one* relation. All processes (including the gather root process) send data items located in their send buffers to the destination buffer located in the root node. Unlike its counterparts, such as *scatter* and *reduce* operations, it has not been receiving much attention in the past. This may be because of amongst all popular collective communications, gather operation has the *simplest* communication structure. Essentially, distinct messages of all participating nodes are worthless to other except the collecting root. No matter how we organize the communication path, the exact amount of data still have to go through to the receiver. Thus, the basic bottleneck of this communication event would be the contention at the receiver end. As our cluster model consists parameters which reflect contention/congestion situation, we are particular interested in utilizing those information in designing algorithms.

The Gather Problem

The *Gather* problem is defined as finding a schedule of communication in the shortest possible time among *p* processors (indexed as P_0 to P_{p-1}), such that initially, each processor P_j, where $j > 0$, has a collection of *i* data items, all destined for processor P_0, with the operation ends with P_0 contains *p, i* data items in the receive buffer.

The communication pattern of the gather operation looks as if it is just the reverse of the scatter operation. However, in reality, we wouldn't implement it this way. As mentioned in [2], the major difference is the status of those processors during the communication. In the case of the scatter operation, root is active and others are passive, while in gather operation, they are active and the root is passive. Hence, the communication schedule of the scatter operation can be coordinated by a centralized means - the scatter root, and the schedule would be more structured. On the contrary, the communication pattern of the gather operation is usually unstructured, as coordination between processors is usually not imposed.

At the start of the event, there are $p' = p\text{-}1$ active senders. The data streams flow from different input ports of the global router and compete for the same outgoing port. As only one frame can be served at a time, the rest have to be buffered in the shared memory of the router. This single stream of outbound flow is the first bottleneck candidate of the whole event. Frame by frame, packets are pumping out through the outgoing port to the receiving NIC, which in turn delivers the arrived packets to their corresponding destination buffers. For the gather operation, this is not necessary the end of the event. Since data are coming from different sources, they are most likely being delivered to different segments of memory. The receiving process upon reception of any messages has to *move/copy* them to their final destination – a contiguous segment of memory. With this scenario, the data reception and movement become another candidate of the bottleneck analysis.

Now we observe that there exists a critical bottleneck in the receiver end, and this is the slowest part of the whole communication event. An apprehensive idea on improving the performance is by minimizing the cost induced by this critical bottleneck stage. As we cannot *compress* the data as if in the case of the *reduce* operation, data must be delivered to the receiver end packet-by-packet till the end. Thus, we argue that the optimality of the gather operation must be one with the following senses: the gather root receives no redundant packets and the data flow through the bottleneck stage continuously.

Definition 1 Of the gather operation, the communication cost is considered to be minimal if and only if it satisfies the following Restrictions:

Restriction 1 The senders only send out their data items at most once and the receiver receives no duplicated items.

Restriction 2 The bottleneck stage of the whole communication event should be full, or, in other words, no *bubble* exists in this pipeline stage.

Lemma 1 There is an optimal algorithm to achieve minimal communication cost for the gather operation that complies with the above Restrictions.

Proof: Let *AlgG* be an optimal algorithm for the gather problem, and it takes $T_G(P)$ units time to finish. If Restriction one is violated, the gather root may receive a particular item more than once, so it has to discard those extra copies. Thus, extra workload induces unnecessary overhead (e.g. data reception and movement). By removing these extra copies, such as, the senders only send out their data items at most once, the new algorithm complies with this Restriction though runs no longer than $T_G(P)$, therefore is optimal. On the other hand, if Restriction 2 is violated, there exist some gaps within the bottleneck pipeline stage. If we modify *AlgG* by ensuring that no bubble appears in the bottleneck stage, the latency induced in the bottleneck stage is reduced. Thus, the new algorithm complies with Restriction 2 and runs no slower than $T_G(P)$ and therefore is optimal. ❑

To satisfy Restriction 2, the optimal algorithm should guarantee that there are enough processors sending out packets to the destination node simultaneously. This is because of with the many-to-one relationship, packets start to cumulate at the staging buffer just before the bottleneck stage. Thus, with sufficient packets available in the buffer together with the ongoing influx of packets, we can guarantee that the bottleneck stage remains full till the end of the communication. To do this, we can simply have all the participating processors sending out their data items simultaneously and without delay. This is the traditional approach and is being used by most of the available parallel programming packages such as MPICH, LAM, CHIMP, PVM3, etc.

5.2. Bottleneck Stage Phenomenon

However, too many processors sending out their packets simultaneously consume enormous amount of system resources, which is limited in the realistic environment, e.g. buffers in the switch and buffers in the receive queue are scare resource. Without appropriate coordination, packets will be lost due to the congestion/overflow condition. Packet loss would trigger higher level reliable protocol to recover the loss, and hence, induces delays and extra overheads. These delays and overheads may result in violation of both Restriction 1 and 2, and drift the whole course away from the optimal. In view of the above observation, an upper bound and a lower bound number of the senders are needed to ensure the algorithm satisfies both Restrictions. The lower bound provides information on the minimum number of simultaneous senders in order to maintain a steady stream of packets to fill up the bottleneck pipeline stage, while the upper bound indicates the maximum number of simultaneous senders are allowed before the resource limit is reached. Based on the parameters provided by our cluster model, we can derive the required upper and lower bounds for this many-to-one collective operation as follows.

Assuming that the communication network is error-free and no software flow control, the only chance of having data loss is by dropping of data message due to saturation of the network resources. We can simplify the bottleneck stage abstractly as a staging buffer associated with input and output pipes. Intuitively, the staging buffer starts queuing up data items when the *departure rate* is slower than the *arrival rate*. So the ratio between the departure rate and arrival rate becomes an indication of congestion. Or in other senses, an indicator of fullness in the departure pipe. Due to the limited size of the buffer, this bottleneck stage cannot sustain long-term congestion, and eventually becomes full. This results in unconditional disposal of arrival items if no more space is left behind. We can encapsulate the above scenario by a simple probability function.

Let the staging buffer be of size ***B*** units, and the arrival and departure rates be ***A*** and ***D*** units per second respectively. When buffer is not full, all incoming items would be buffered and forward later on (assume ***A > D***). At a particular instance, $t = \beta$, we have the fullness condition of the staging buffer, then

$$A \cdot \beta = B + D \cdot \beta \qquad \Rightarrow \qquad \beta = \frac{B}{A - D} \tag{3}$$

Before $t = \beta$, incoming items will not be dropped by the system, the probability of transfer is equal to one. After the saturation, $t > \beta$, newly arrived items can only be accepted and transferred in a probability of $\frac{D}{A}$. Hence, we would expect the transfer ratio of the bottleneck stage in long run becomes

$$\frac{\left(k - \frac{A \cdot B}{A - D}\right) \cdot \frac{D}{A} + \frac{A \cdot B}{A - D}}{k} \Rightarrow \frac{D}{A} + \frac{B}{k} \tag{4}$$

where k is the total number of incoming data items through the bottleneck stage. In summary, if $\frac{D}{A} + \frac{B}{k} \geq 1$ then the transfer ratio is one, else the transfer ratio is $\frac{D}{A} + \frac{B}{k}$.

Lower Bound -- Ga_l

When the number of simultaneous senders is less than this lower bound value, the critical bottleneck stage could not fill up with data. This situation means that the departure rate is faster than the arrival rate. Therefore, the $\frac{D}{A}$ ratio correlates to the lower bound value for our bubble-free bottleneck stage. More precisely, let's analyze the bottleneck stage at the switching hardware, this lower bound value can be derived as

$$Ga_l \;\Rightarrow \frac{D}{A} \leq 1 \Rightarrow \frac{1/g_r}{p' \cdot 1/g_s} \leq 1 \Rightarrow \frac{g_s}{p' \cdot g_r} \leq 1 \Rightarrow \frac{g_s}{g_r} \leq p' \;\Rightarrow \left\lceil \frac{g_s}{g_r} \right\rceil \tag{5}$$

With $Ga_l \leq \left\lceil \frac{g_s}{g_r} \right\rceil$, we can guarantee that $\frac{D}{A} < 1$ and data items would start to queue up at the staging buffer which in turn, provides a full stream of data to fill up the bottleneck pipe.

Upper bound -- Ga_u

As we have shown that on long run, to avoid data loss due to saturation, we have to keep the transfer ratio to be as close as one. Hence we cannot have unlimited number of simultaneous senders as this would eventually flood the bottleneck stage and cause overflow. So the optimal value can be determined at the situation where the transfer ratio equals to one. With the bottleneck stage appears at the switching hardware, this upper bound value can be derived as

$$Ga_u \;\Rightarrow \frac{D}{A} + \frac{B}{k} \geq 1 \Rightarrow \frac{g_s}{p' \cdot g_r} + \frac{B}{p' \cdot i} \geq 1 \Rightarrow \frac{g_s}{g_r} + \frac{B}{i} \geq p' \;\Rightarrow \left\lfloor \frac{g_s}{g_r} + \frac{B}{i} \right\rfloor \tag{6}$$

This upper bound value, in fact, gives us performance hints to fully utilize the network resources. If we have an infinite buffer, the upper bound value becomes unbound, all incoming traffics can be safely buffered and delivered to their destinations. Or, if we know that the total amount of data items to be collected (k) is smaller than the available buffer (B), the upper bound value would then bound by p' only.

5.3. Our gather Algorithm

Any value w lies between the lower and upper bounds could be a candidate to satisfy our optimality definition of the gather operation. To search for the best value of w, we have the following formula,

$$w = \text{MAX}\left[\{x : Ga_l \leq x \leq Ga_u \wedge \text{REMAINDER}(p', x) \geq Ga_l\}\right]. \tag{7}$$

This returned value guarantees that at any particular instant, the sender window w is at least greater than or equal to the lower bound value Ga_l, such that the bottleneck stage is bubble-free with no packet loss during the communication event. Our algorithm is just a simple extension of the traditional gather operation enforces with processors coordination, and it works by restricting the number of active senders equal to w at a particular instant. When start, if a cluster node is within the active sender window, they are free to send all their data items to the gather root. Or else, they have to wait for the

```
if my index j > 0
    if B > p' · i
        sends all my data items to P_0 immediately
    else
        if j ≥ w
            sends all my data items to P_0 immediately
            send a startup signal to processor P_{j+w} if j + w < p
        else
            wait for startup signal from processor P_{j-w}
            sends all my data items to P_0 immediately
            send a startup signal to processor P_{j+w} if j + w < p
        endif
    endif
else I am the gather root P_0
    receive p' · i data items from P_1 to P_{p-1}
endif
```

Figure 4 The pseudocode of the optimal gather algorithm

startup signal before they are allowed to transmit their data. To send the startup signal, we have two candidates, the gather root and those active senders. We opt to use those active senders for two reasons. First, it is hard to detect when an active sender has finished its job at the gather root, as too late to activate a waiting sender may cause unnecessary *bubbles* formation in the bottleneck pipe. Second, this relieves the workload of the gather root, as it is already overloaded by the influx of data packets. Figure 4 contains the pseudocode that summaries our gather algorithm.

5.4. Execution Time Analysis

If this algorithm conforms to our optimality Restriction, we should expect the lower bound of the execution time be confined to $T(p,i) \geq (p-1) \cdot i \cdot D$, the time engaged in transmitting all data items through the critical bottleneck stage. Hence, the overall execution time of our algorithm satisfies,

$$T(p,i) \geq C_1 + (p-1) \cdot i \cdot D + C_2 \tag{8}$$

where C_1 is defined as the arrival time of the first data item to the critical bottleneck stage, and C_2 is the service time of the last data item from the exit of the critical bottleneck stage to the final destination buffer.

5.5. Experimental Results

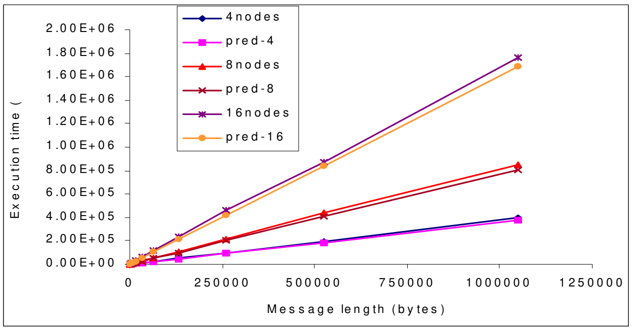

Figure 4 Measured and estimated execution times with our cluster model for our optimal gather algorithm

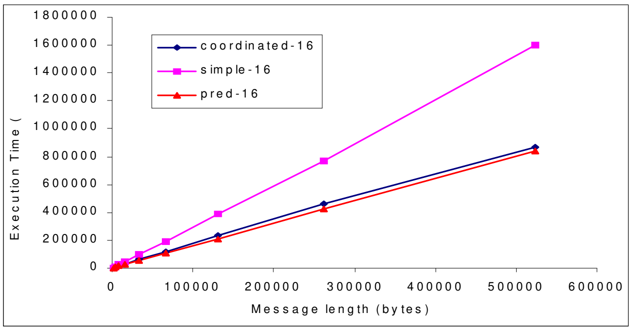

Figure 5 Comparison on two gather algorithms with the prediction

To verify the analytical and prediction accuracy of our cluster model, we implemented the above gather algorithm on our experimental platform. We have measured the running time of the operation ranging from $p=4,8,16$ and message length from 1K to 1M bytes. The results are shown in figure 5 where both measured and predicted times are plotted. This figure shows that the prediction closely match the measurement with the average prediction error of 8%. The maximum error is 35% which is located at $p=4$, and message length of 1KB. This is mostly caused by violation of Restriction 2, as the number of participating nodes and the message length is not large enough.

Figure 6 shows the comparison of the measured run-times of two gather algorithms. The *coordinated* algorithm is the one that we have proposed and the *simple* algorithm is the traditional uncoordinated algorithm which used by most of those available parallel packages. It is clearly shown that without proper communication scheduling, limited resources are overloaded which results in suffering of unnecessary delays. This situation is getting worse when we increase the number of participating nodes or increase the message length.

6. CONCLUSIONS

In this paper, we have introduced a simple communication model that allows precise predictions of communication costs on the cluster environment. The model exposes all the crucial performance characteristics of the system by a set of parameters. We explicitly introduce the contention issue into our parameters, e.g. the *transfer phase* and the *network buffer* parameters. And based on this set of parameters, we can perform analysis on any target architecture-application pair. We used the popular *log N* broadcast operation to show the delay caused by contention during simultaneous send and receive operations, and this delay can only be explained by our model. To show the analytical power of our model, we have developed an optimal gather algorithm with the ability to avoid contention loss. By restricting the number of active senders within the upper and lower bound, we have shown that our algorithm can significantly reduce the communication cost.

7. REFERENCES:

[1] A. Bar-Noy and S. Kipnis. Designing Broadcasting Algorithms in the Postal Model for Message-Passing Systems. In *Proceedings of the ACM Symposium on Parallel Algorithms and Architectures*, pp. 11-22, June 1992.

[2] S.N. Bhatt, G. Pucci, A. Ranade, and A.L. Rosenberg. Scattering and Gathering Messages in Networks of Processors. In *IEEE Trans. on Computers*, C-42:938-949, 1993.

[3] D.E. Culler, R.M. Karp, D.A. Patterson, A. Sahay, K.E. Schauser, E. Santos, R. Subramonian and T. von Eicken. LogP: Towards a Realistic Model of Parallel Computation. In *Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, May 1993.

[4] T. Heywood and C. Leopold. Models of Parallelism. Technical Report CSR-28-93, Department of Computer Science, University of Edinburgh, 1993.

[5] Intel Corp. Scalable Stacking Technology. (http://www.intel.com/network/technologies/scalable_stacking.htm)

[6] R.M. Karp, A. Sahay, E.E. Santos and K.E. Schauser. Optimal Broadcast and Summation in the LogP Model. In *5th Symp. on Parallel Algorithms and Architectures*, June 1993.

[7] C.M. Lee, A. Tam, and C.L. Wang. Directed Point: An Efficient Communication Subsystem for Cluster Computing. In *The International Conference on Parallel and Distributed Computing Systems (IASTED)*, October 1998.

[8] W.F. McColl. The BSP Approach to Architecture Independent Parallel Programming. Oxford University Computing Laboratory, March 1995. (ftp://ftp.comlab.ox.ac.uk/pub/Documents/techpapers/Bill.McColl/p7.ps.Z)

[9] J.M. Nash, P.M. Dew, and M.E. Dyer. Scalable and portable computing using the WPRAM model. In *Kara, M, Davy, J R, Goodeve, D M & Nash, J M (editors), Abstract Models for Parallel and Distributed Computing*, pp.47-62.-IOS Press, 1996.

[10] R. Subramonian and N. Venkatasubramanyan. Optimal Broadcast in a Distributed Memory Model of Parallel Computation.

[11] L.G. Valiant. A Bridging Model for Parallel Computation. *Communication of the ACM*, August 1990, Vol. 33, No. 8, pp. 103-111.