Reduction Optimization in Heterogeneous Cluster Environments

Pangfeng Liu
Department of Computer Science
National Chung Cheng University
Chiayi, Taiwan, R.O.C.
pangfeng@cs.ccu.edu.tw

Da-Wei Wang
Institute of Information Science
Academia Sinica
Nankang, Taipei, R.O.C.
wdw@iis.sinica.edu.tw

Abstract

Network of workstation (NOW) is a cost-effective alternative to massively parallel supercomputers. As commercially available off-the-shelf processors become cheaper and faster, it is now possible to build a cluster that provides high computing power within a limited budget. However, a cluster may consist of different types of processors and this heterogeneity complicates the design of efficient collective communication protocols. For example, it is a very hard combinatorial problem to find an optimal reduction schedule for such heterogeneous clusters. Nevertheless, we show that a simple technique called slowest-node-first *(SNF) is very effective in designing efficient reduction protocols for heterogeneous clusters. First, we show that SNF is actually an approximation algorithm with competitive ratio two. In addition, we show that SNF does give the optimal reduction time when the cluster consists of two types of processors, and the ratio of communication speed between them is at least two.*

1 Introduction

Network of workstation (NOW) is a cost-effective alternative to massively parallel supercomputers [2]. As commercially available off-the-shelf processors become cheaper and faster, it is now possible to build a PC or workstation cluster that provides high computing power within a limited budget. High performance parallelism is achieved by dividing the computation into manageable subtasks, and distributing these subtasks to the processors within the cluster. These off-the-shelf high-performance processors provide a much higher performance to cost ratio so that high performance cluster can be built inexpensively. In addition, the processors can be conveniently connected by industry standard network components.

In parallel to the development of inexpensive and standardized hardware components for network of workstations, system softwares for programming on NOW are also advancing rapidly. For example, the *Message Passing Interface* (MPI) library has evolved into a standard for writing message-passing parallel code [1, 4, 5]. MPI programmers use a standardized high-level programming interface to exchange information among processes, instead of a native machine-specific communication library. MPI programmers can write highly portable parallel codes and run them on any parallel machine that has MPI implementation.

Most of the literature on cluster computing emphasizes *homogeneous* clusters – clusters consisting of the same type of processors. However, we argue that heterogeneity is one of the key issues that must be addressed in improving performance of NOW. First it is always the case that one wishes to connect as many processors as possible to increase parallelism. Despite the increased computing power, the scheduling management of such a *heterogeneous network of workstation* (HNOW) becomes complicated since these processors will have different communication performance from one another. Second, since most of the processors that are used to build a cluster are commercially off-the-shelf products, they will very likely be outdated by faster successors before they become unusable. Very often a cluster will consist of "leftovers" from the previous installation, and "new comers" that are recently purchased. The issue of heterogeneity is both scientific and economic.

Any workstation cluster requires efficient collective communications [3]. For example, a scatter operation distributes input data from the source to all the other processors for parallel processing, then a global reduction operation combines the partial solutions obtained from individual processors into the final answer. The efficiency of these collective communications will affect the overall performance, sometimes dramatically.

Heterogeneity of a cluster complicates the design of efficient collective communication protocols. When the processors send and receive messages at different rates, it is difficult to synchronize them so that the message can arrive at the right processor at the right time. On the other hand, in a homogeneous NOW every processor requires the same amount of time to transmit a message. For example it is straightforward to implement a reduction operation on P processors in $\log P$ steps. In a heterogeneous environment it is no long clear how we should proceed to complete the same task.

This paper shows that a simple heuristic called *slowest-node-first* (SNF) is very effective in designing reduction protocols for heterogeneous cluster systems. Despite the fact that SNF heuristic does not guarantee the optimal reduction time, we show that SNF is actually an approximation algorithm with competitive ratio two. In addition, we show that SNF does give the optimal reduction time when the cluster consists of only two types of processors, and the ratio of communication speed between them is at least two.

The rest of the paper is organized as follows. Section 2 describes the communication model in our treatment of reduction problem in HNOW. Section 3 describes the concept of the earliest possible schedule and Section 4 describes the slowest-node-first heuristic for reduction in heterogeneous cluster. Section 5 states the main results and Section 6 concludes.

2 Communication Model

The communication model of a heterogeneous environment is defined as follows. The system consists of n processors $\{p_0, ..., p_{n-1}\}$, each is capable of direct point-to-point communication to one another. A processor p_i is characterized by its transmission time $t(p_i)$, i.e. the time it takes for p_i to send a message to any other processor.

In order to make the communication model realistic, we further assume that two communications cannot overlap, i.e. neither the sender nor the receiver of an ongoing communication can engage in any other communication at the same time. This assumption is based on that in practice most of the processors in a clusters only have limited resource for communication.

We define the reduction problem with the communication model we just described. Suppose each processor has a unit of information, and we would like to combine all these informations into the final answer, how do we schedule the processors so that the reduction takes the least amount of time? For a homogeneous system a simple tree algorithm can combine all the information in $\log n$ rounds. However, due to the variance in communication speed in a heterogeneous environment, this algorithm cannot guarantee minimum reduction time.

We formally define the reduction as a scheduling problem. We observe that during the reduction process exactly $n-1$ messages are sent by $n-1$ different processors. As a result the slowest processor should only receive message and compute the final result. Based on this observation, we define a reduction schedule as a function from a processor to the time that it start sending its message. Formally we define the schedule function s so that for any processor p, p starts and completes sending its message at time $s(p)$ and $s(p)+t(p)$ respectively. The interval $[s(p), c(s,p)]$ is defined as the *transmission window* of p, where $c(s,p) = s(p)+t(p)$ is the *completion time* of p under the schedule s.

Due to the constraint that a processor can only participate a single communication at any given time, we must distinguish valid schedule from invalid ones. To formalize this constraint, we define two sets of processors for any schedule at any given time. Let $A(s,t)$ be the number of processors that are still actively sending their messages, and $C(s,t)$ be the number of processors that have completed sending their messages at time t under schedule s.

$$\begin{aligned} A(s,t) &= |\{p_i : s(p_i) \leq t < c(s,p_i)\}| \\ C(s,t) &= |\{p_i : t \geq c(s,p_i)\}| \\ 2A(s,t) + C(s,t) &\leq n \end{aligned} \quad (1)$$

A schedule function s is *valid* if and only if inequality 1 is true, which says that that at any time t, the total number of senders and receivers, and those processors that have sent out their messages, should not be more than the number of processors. Figure 1 show a valid schedule.

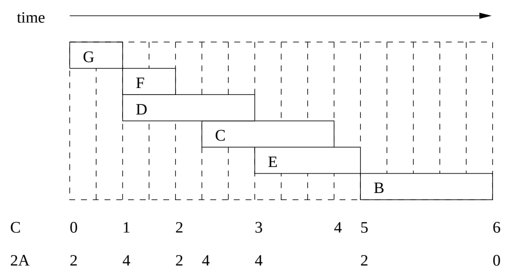

Figure 1: An valid schedule example showing the A and C functions. The system consist of seven processors from A to G. The transmission time for the processors are 10, 5, 5, 5, 4, 2, and 2.

3 Earliest-Possible Schedule

This section describe a technique called *earliest possible* scheduling that can "normalize" all the possible valid schedules. We use this canonical form to simplify the discussion of finding an optimal schedule.

An earliest possible (EP) schedule is one in which all the communications are initiated *as early as possible*. A new communication can be initiated as soon as the number of *free processors* reaches 2 – one as the sender and the other as the receiver. Let $F(s,t)$ denote the number of processors that we are free to use to initiate a new communication at time t under schedule s. That is, $F(s,t) = n - CA(s,t) - C(s,t)$, where $CA(s,t) = |\{P_i | s(P_i) < t < s(P_i) + t(P_i)\}|$ is the number of processor that are in the middle of sending and receiving messages. $CA(s,t)$ is different from $A(s,t)$, which includes processors that will start sending and receiving message at time t. As a result $\lfloor n/2 \rfloor$ processors will start sending messages at time 0, and the rest will start sending messages as soon as the number of free processors reaches two, as illustrated by Figure 2. The algorithm EP will assign non-decreasing start time to the processors in the order they appear in the input processor sequence P.

```
Algorithm EP(P)
{
  i = 1; time = 0; free = n;
  Active = empty set; Complete = empty set;
  while (i ≤ n-1) {
    do while (free ≥ 2) {
      set the start time of the ith processor in P
      to time;
      add the ith processor in P into Active;
      i = i + 1;
      free = free - 2;
    }
    find the set of processors p that has the smallest
    completion time in Active;
    time = the completion time of p;
    move p from Active to Complete;
    free = free + the number of elements in p ;
  }
}
```

Figure 2: The earliest possible scheduling algorithm.

By definition an earliest possible schedule s has the following properties. See Figure 3 for details.

1. $n - 1 \leq 2A(s,t) + C(s,t) \leq n$.
2. For any processor p that $s(p) > 0$, there exist another processor q such that $s(p) = c(s,q)$.

Note that we can convert any valid schedule into an EP schedule by moving the transmission windows earlier, without increasing the total time. As a result EP schedule servers as a canonical form in which we will limit our search of an optimal schedule. Figure 3 shows the earliest possible schedule converted from Figure 1.

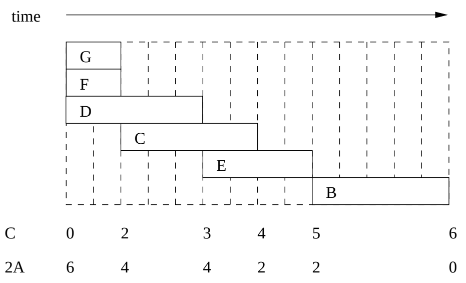

Figure 3: The EP schedule converted from the schedule in Figure 1

3.1 Sender Dependency and Destination Assignment

In Figure 1 we only show the senders of the reduction communication. For for any EP schedule, it is possible to specify all the destinations without violating the constraint that a processor cannot participate more than one communication simultaneously.

Lemma 1 *For any earliest possible schedule we can assign the destinations for all processors so that no two overlapping transmission windows have any sender or receiver in common.*

Proof. First we establish the dependency among the processors. Note that in algorithm EP it requires two free processors to start a new transmission. As a result we define the *predecessors* of a sender processor to be the two senders that must complete *before* the new transmission can start (when the number of processor is odd , one of the processor will have only one predecessor). This dependency forms a binary tree among all sender processors. See Figure 4 for an illustration.

Now we select destinations for all senders. First we assign the slowest processor as the destination of the last send. Then for a sender/receiver pair s and r, we assign s and r to be the destination of the two predecessors of s. Since both predecessors complete sending their messages before s starts, no overlapping transmission windows have common processors. ■

Figure 4 shows the destination for the schedule in Figure 3. The assignment is not unique since we can assign s and r to either of the two predecessors.

4 Slowest-Node-First Schedule

We introduce a simple scheduling method called *slowest-node-first* (SNF) for the reduction problem. SNF sorts the processors in non-increasing order, and gives the sorted sequence $P = (p_1, p_2, ...p_{n-1})$, $t(p_i) \geq t(p_j)$ if $i < j$, to algorithm EP. This simple technique

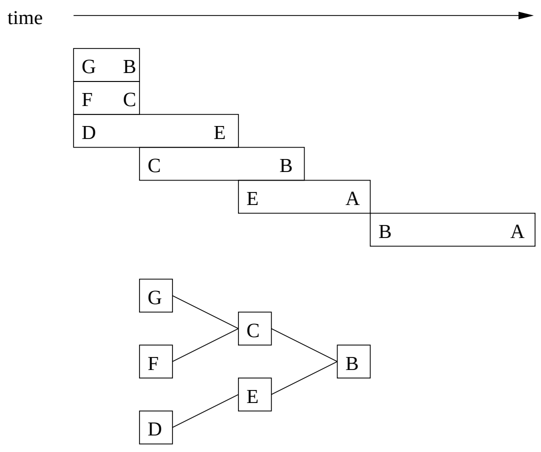

Figure 4: The destinations for all senders in Figure 3 and the predecessor dependency tree.

is very effective in obtaining a good reduction schedule. Figure 5 shows the SNF schedule from the cluster in Figure 3.

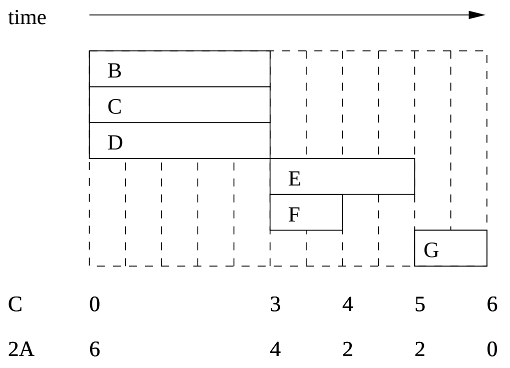

Figure 5: SNF schedule from the cluster in in Figure 1 and 3.

The rationale of having the slowest processors to send messages first is as follows. At the beginning of the reduction process, we would like to overlap as many communication windows as possible. Intuitively we let all the slow processors send first so that they will overlap with each other, instead of having to wait for each other at the end of the reduction.

5 Main Results

We begin with an exchange lemma that clarifies our intuition that slow processors should send first.

Lemma 2 *Let s be a valid schedule and processor p starts right after processor q ends, i.e. $s(p) = c(s,p)$. If $t(p) > t(q)$ then we can exchange p and q so that the*

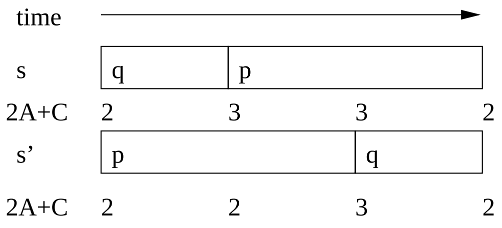

Figure 6: An illustration on the contribution to the sum of C and $2A$.

modified schedule s' in which $s'(p) = s(q)$ and $s'(q) = c(s', p)$ is also valid.

Proof. From Figure 6 we observe that the contribution of p and q to the sum of C and A functions is always higher in s than in s', therefore if s can satisfy Inequality 1, so can s'. ■

From Lemma 2 it follows that in the search of an optimal reduction schedule we can neglect those schedules that have a slower sender p with a faster predecessor q. There are two cases to consider. First if $s(p) = c(s,q)$ then by Lemma 2 we can switch p and q. If there is a gap between the transmission windows of p and q, then we can delay the window of q until it touches p's window since there is no new transmission initiated between $s(p)$ and $s(q) + t(q)$.

Corollary 1 *There exists an optimal schedule such that the predecessors of every senders have an equal or higher transmission time than the sender.*

5.1 Competitive Ratio

We now consider a special class of clusters (called *power 2* clusters) in which the transmission time of every processor is a power of 2. Without lose of generality we assume that the fastest processor has a transmission time of 1, and the slowest one has 2^k. We show that SNF gives an optimal scheduling for all power 2 clusters.

Theorem 1 *The slowest-node-first method gives an optimal schedule for all power 2 clusters.*

Proof. We show that every schedule for a power 2 cluster can be converted into the SNF schedule without increasing the total reduction time. We reschedule all the slowest processors as early as possible, until there is no faster processor ahead of them. Then we reshcdule the second slowest processors the same way, and repeat this rescheduling until the final schedule is the same as SNF.

When the transmission time of every processors is a power of 2, we claim that the start time of every processor is a multiple of its transmission time. Since every processor has a predecessor that ends when it starts, we locate a "chain" of predecessors all the way back to time 0. In addition, every one of these predecessors has a transmission time of equal or larger power of 2. Therefore the start time, which is the sum of the transmission time of these predecessors, is also a multiple of its transmission time.

Consider a slowest processor p, and the set F of faster processors that starts before p. Let q be the processor completes last in F. If q finishes right when p starts, then by Lemma 2 we can exchange p and q. If there is a gap between $s(p)$ and $s(q) + t(q)$, then any processors starts within this gap must end within this gap as well since all the processor must start and end at the multiple of its transmission time. Therefore q will not be the processor with the largest completion time in F. Similarly we argue that any processor completes in that gap must also start in that gap, therefore we can safely delay the transmission window of q so that q ends when p starts. By Lemma 2 we can then exchange p and q and the theorem follows. ∎

Theorem 2 *The slowest-node-first schedule has a total reduction time no greater than twice of the time of an optimal schedule for* all *clusters.*

Proof. Let s be an optimal schedule for a cluster P. Without lose of generality we assume that the fastest processor in P has a transmission time 1. We first convert P into a power 2 cluster by increasing the transmission time of every processor p to $2^{\lceil \log t(p) \rceil}$. We call this new cluster P'.

We argue that there exists a schedule s' for P' in which every processor p starts at time no later than $2s(p)$. The claim follows from a simple induction that if every processor p in P' waits for the two predecessors q and r defined by s, then it can start at time no later than $2s(p)$, since both p and q starts no later than $2s(q)$ and $2s(q)$, and their transmission time at most double in P'.

Now we have constructed a new schedule s' for P' that has a reduction time at most twice of s. Since s' is a schedule for a power 2 cluster, its reduction time will be no less than the optimal schedule s^* produced by SNF for P'. Finally the SNF schedule s'' on P is at least as fast as the SNF schedule s^* for P', since every processor starts earlier in P than in P' because both of its two predecessors can start earlier in P, and sends message faster. Therefore the reduction time of s'' is no more than s^*, which in turn is no more than twice of s. ∎

5.2 Two Types of Processors case

We prove that SNF gives the optimal reduction time when the cluster consists of two types of processors and the ratio of communication speed between them is at least *two*.

Lemma 3 *Let s be a valid schedule, p be a slow processor and q_1, q_2, q_3 be fast processors. If $s(p)$ is within the transmission window of q_1, q_1 stars right after q_2, and q_2 starts right after q_3, then there exists a schedule s' with following properties.*

1. *$s'(p) = s(q_3)$, i.e. process p starts sending message in s' the time process q_3 starts sending message in s.*

2. *$s'(q_3) = s(p)$, i.e. process q_3 starts sending messages in s' the time process p starts sending message in s.*

3. *Process q_1 starts right after q_2 and q_1 ends in s' at the same time as p ends in s.*

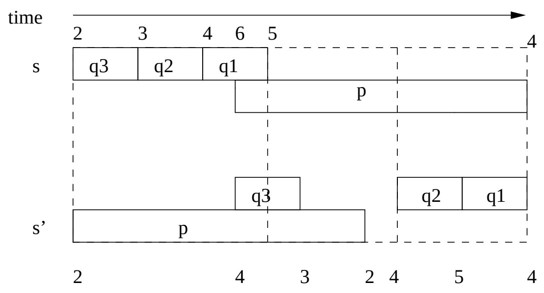

Figure 7: Rearrange the communication windows of p and q_1, q_2, q_3.

Proof. From Figure 7 we observe that $2A(s, s(p)) + C(s, s(p)) = 6$ because p is at least twice as slow as q_1, and $s'(q_2) < c(s', p)$, i.e. p ends before q_2 starts in s'. The theorem follows since at any time t the value of $2A + C$ is always equal or higher in s than in s'. ∎

By Lemma 2 there exists an optimal schedule in which no slow processor will start *right after* a fast processor. Therefore, we can assume for every slow processor, its predecessors are slow processors as well. The starting time of every slow processor is a multiple of its transmission time, and we can layer the slow processors according to their starting time as follows.

Definition 1 *Layer one processors are the set of slow processors starting at time 0. Layer $i + 1$ processors are the set of slow processors starting right after layer i processors.*

Theorem 3 *SNF is optimal when there are only two types of processors and the ratio of communication speed between them is at least two.*

Proof. Assume that there exists an optimal solution in which there exits a slow processor that starts later than a fast processor. Let p be the one in the earliest layer among such slow processors, and q be the one that has the latest completion time among all those fast processors before p. We show that it is always possible to reschedule p so that it starts earlier than q without increasing the total time.

There are three cases to consider – the processor p can start *before*, *at*, or *after* the processor q ends. For the second case we apply Lemma 2 and switch p and q. For the third case, since q finishes last among all the fast processors starting before p, there will be no processor starting after q ends and before p starts. As a result we can delay the transmission window of q so that q ends right when p starts, then we apply Lemma 2 to move p earlier.

We now consider the first case. Since every processor has a predecessor that ends when it starts, we locate a "chain" of predecessors for q all the way back to time 0. For the processor q we need at least two levels of predecessors and both of them must be fast ones, since the ratio of communication speed is at least two and the slow processors must start at a multiple of its transmission time. By Lemma 3 we find a new schedule when p starts before q. We repeatedly reschedule the slow processors p earlier until no fast processor before it, and the theorem follows. ■

Figure 8 shows a counterexample that SNF always gives optimal reduction time. This cluster has 4 slow processors with transmission time x, and 8 fast processors with transmission time 1. The alternative schedule requires $2x + 1$ time when $1.5 \leq x < 2$, or 4 when $1 < x < 1.5$. In contrast SNF requires $x + 3$ time for both cases, and has a longer reduction time for for all x between 1 and 2. As a result the bound of two in Theorem 3 is tight.

6 Conclusion

This paper shows that the slowest-node-first scheduling is very effective in designing reduction communication protocols for heterogeneous cluster systems. We show that SNF is actually an approximation algorithm with competitive ratio two. In addition, we show that SNF does give the optimal reduction time when the cluster consists of two types of processors, and the ratio of communication speed between them is at least two.

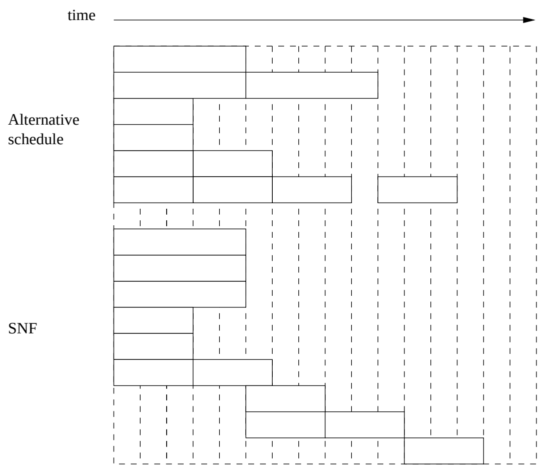

Figure 8: A counterexample that SNF always gives optimal reduction time.

It will be interesting to extend this technique to other communication protocols and models. In a more practical and complex model the communication time may be a function of both the send and the receiver. Also it is important to investigate the possibility to extend the analysis to similar protocols like parallel prefix, all-to-all reduction, or multicast. These questions are very fundamental in designing collective communication protocols in heterogeneous clusters, and will certainly be the focus of further research in this area.

References

[1] *Message Passing Interface Forum.* Mar 1994.

[2] T. Anderson, D. Culler, and D. Patterson. A case for networks of workstations (now). In *IEEE Micro*, Feb 1995.

[3] M. Banikazemi, V. Moorthy, and D.K. Panda. Efficient collective communication on heterogeneous networks of workstations. In *Proceedings of International Parallel Processing Conference*, 1998.

[4] J. Bruck et al. Efficient message passing interface(mpi) for parallel computing on clusters of workstations. *Journal of Parallel and Distributed Computing*, Jan 1997.

[5] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of the mpi: a message passing interface standard. Technical report, Argonne National Laboratory and Mississippi State University.