An Adaptive Communication System for Heterogeneous Network Computing

Ilkyeun Ra
Department of Electrical Engineering and Computer Science
Syracuse University
Syracuse, NY 13244
ikra@ecs.syr.edu

Salim Hariri
Department of Electrical and Computer Engineering
The University of Arizona
Tucson, AZ 85721
hariri@ece.arizona.edu

Caulgi Raghavendra
Department of EE-Systems
University of Southern California
Los Angeles, CA 90089
raghu@halcyon.usc.edu

Abstract

The Adaptive Communication System(ACS) provides programmable communication, control and management services that can be optimized to meet the quality of service requirements of parallel and distributed applications. ACS supports adaptive and scalable communication services that select the appropriate multicast/broadcasting algorithms for a given class of applications. These algorithms take into consideration both the application requirements and the load of computing and communication systems.

In this paper, we present an architecture of an Adaptive Communication System (ACS) that provides applications with programmable communication, control, and management services that can be adopted dynamically to maximize application performance at runtime. We overview the ACS architecture and then describe our approach to implement the ACS group communication services. We introduce two procedures (Resource Aware procedure and Application Aware procedure) to build the appropriate multicast tree that takes into consideration both the characteristics and load conditions of ma chines as well as the group communication patterns of a given application. We develop analytical techniques and new metric measures to characterize and quantify the performance of a multicast tree. We also present our preliminary performance results that show significant performance gain can be achieved from using ACS multicast algorithms.

1. Introduction

With the recent advances in computer network technologies, we have seen a rapid development of high performance distributed systems that utilize heterogeneous resources that are geographically dispersed across a wide area network (e.g., Gridcomputing, Meta-computing Environments [7, 8, 14]). Due to the commodity nature of gridcomputing, the resources in the environment includes a wide range of computing systems that vary from high performance computers (e.g., supercomputer, special purpose parallel machine) to desktop computers/workstations and networks that are wired, wireless and have a wide range of transmission speeds (e.g., from few Kbps to Gbps).

However, most of the current message passing systems and tools [3, 4, 5, 6, 9, 10, 11, 12, 13, 15] do not provide dynamic and programmable communication services that can be selected at runtime to meet the requirements of a wide range of network centric applications (e.g, Grid applications, distributed applications). Consequently, the performance of the applications degrade significantly when the collective communication services required by these applications are not well supported by the underlying message passing systems and/or tools. There are few research projects that aim at developing resource scheduling algorithms that utilize the heterogeneity of networks and computing resources [1, 2].

In this paper, we overview the ACS architecture and communication services, and describe two adaptive communication resource scheduling algorithms: *Resource Aware scheduling, and Application Aware scheduling* to build the appropriate multicast tree that takes into consideration both the type of machines as well as the collective communication pattern. We introduce analytical techniques and new metric measures to characterize and quantify the performance of a multicast tree that takes into consideration both the type of machines as well as the group communication pattern. Our preliminary performance results show the performance gain that can be achieved from using ACS multicast algorithms.

The organization of this paper is as follows. In Section 2, we give a brief overview of the ACS architecture. In Section 3, we discuss the ACS resource aware multicasting communication services and algorithms. In Section 4, we present the ACS application aware multicasting services and algorithms. In section 5, we show experimental results and evaluation of the ACS multicasting algorithm. In section 6, we provide summary and concluding remarks.

2. ACS Architecture Overview

The main goal of the ACS architecture (see Figure 1) is to provide the communication services and protocol functions that can satisfy efficiently the requirements of a wide range of network-centric applications.

The ACS resides between the application and the transport layer and consists of two planes: Data Communication Plane(DCP), and Control and Management Plane(CMP). The *DCP* provides a rich set of communication services that allows applications to cooperate and exchange information. These communication services include *Point-to-point communication*,*Group communication* services, and *Multiple communication interfaces* such as Socket Communication Interface (SCI), ATM Communication Interface (ACI), and Wireless Communication interface (WCI). The *CMP* supports the appropriate control and management functions required by applications such as Error control (EC), flow control (FC) , Compression control (CP), security control (SC) and application Management(AM). For each application, we establish two Connections: one connection to be used by DCP services, while the second one is dedicated for control and management functions. For example, while an application is running, the control and management connection is maintained and monitored by one or more agents. These agents can intervene and change the flow control, error control, compression algorithm if the current system state and monitored by one or more agents. These agents can intervene and change the flow control, error control, compression algorithm if the current system state faults to meet the application quality of service requirements.

Once the application requirements are defined (e.g., performance, fault tolerance), the ACS can then build the appropriate communication services that are appropriate for the application. Furthermore, the ACS agents can use the control and management connection to change the communication protocols and functions at runtime when the network conditions change and require the implementation of different error and flow control mechanisms. In what follows, we describe in further detail the ACS group communication services and evaluate their performance.

3. ACS Collective Communication Overview

Most of collective communication primitives are typically implemented on top of point-to-point communication routines. For example, the multicast operation in PVM has been implemented by repeatedly sending the message to each group member. This implementation approach results in high communication overhead and hindering scalability especially when the group size becomes large. Furthermore, the implementation approach of collective communication routines are rigid and can not be changed at runtime when the network conditions adversely affect their performance.

To overcome those limitation of current collective communication services, the ACS implementation approach is flexible and does support adaptive implementation of the ACS communication services. In what follows, we overview the basic ACS collective communication(group communication).

3.1. ACS collective communication primitives

*int ACS_mcast(char *gname[], ACS_Dtype type, int tag, char *msg, int len);*
- Multicasts a message to the groups specified by gname[] with an multicasting algorithm.

*int ACS_group_create(char *gname, int com_mode, int fc, int ec, int mc, struct QoS);*
- Creates a group named *gname*. Returns the group identifier.

*int ACS_group_join(char *gname);*
- Joins the group specified by *gname*.

*int ACS_group_destroy(char *gname);*
- Destroys the group specified by *gname*.

*int ACS_group_leave(char *gname);*
- Leaves the group specified by *gname*.

*int ACS_group_num_members(char *gname);*
- Returns the total number of members in the group.

*int ACS_bcast(int mode, char *gname[], ACS_Dtype type, int tag, char *msg, int len);*
- Broadcast a message to the all groups members in gname[].

Figure 2. ACS Group Communication Primitives

ACS allows a dynamic formation of groups, any process can dynamically join and/or leave a group. All the communications related to a group are handled by a single group

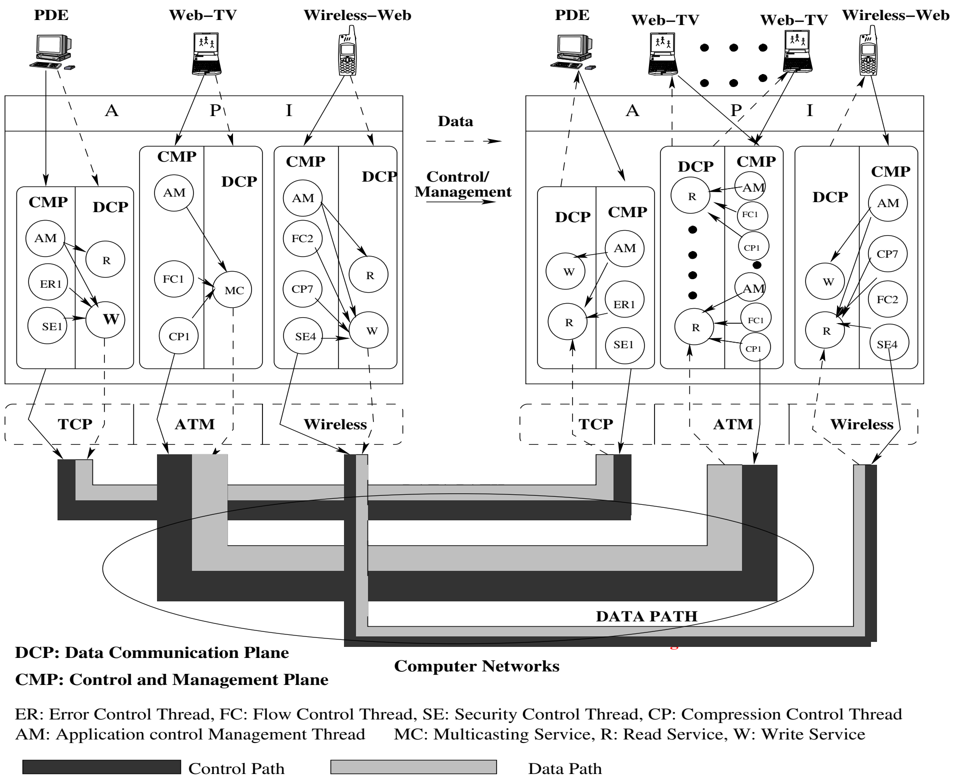


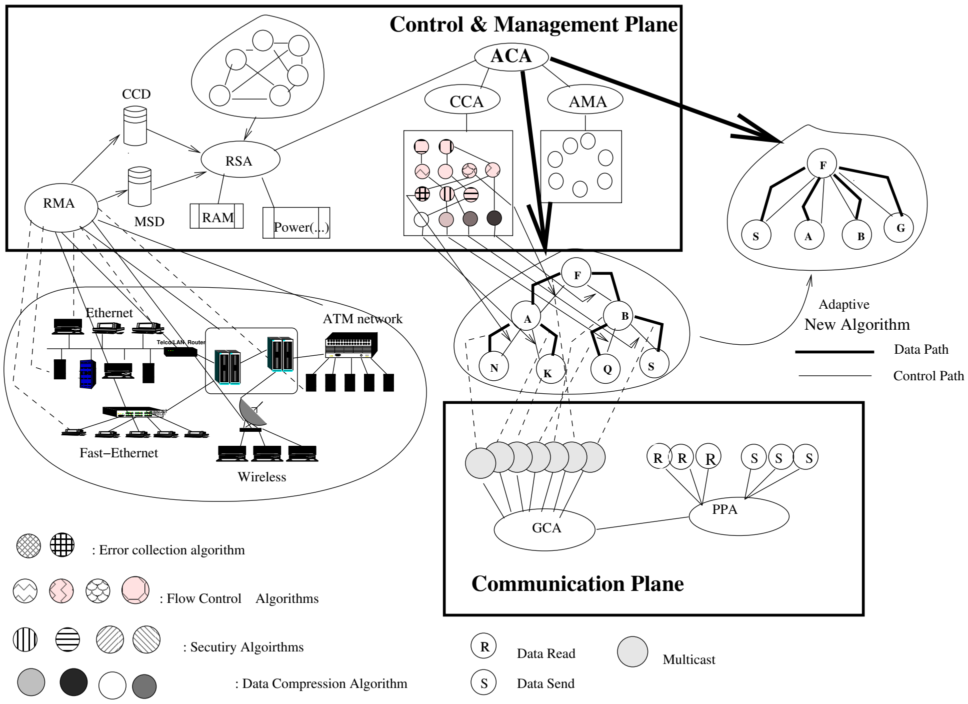
Figure 1. An Adaptive Communication Software Architecture

server. Within each group there is a single group server that is responsible for intergroup and multicasting communications. The default implementation of the ACS multicasting is based on a tree based protocol, which is more efficient than repetitive techniques for large group sizes. The ACS architecture, which separates the data and control & management transfer, allows multicasting operations to be implemented efficiently by utilizing the control connections when transferring status information (e.g., membership change, acknowledgment to maintain reliability, etc.). This separation optimizes the data path and improves the performance of ACS applications. Current implemented ACS group communication primitives are shown in Figure 2.

Figure 3 shows how ACS utilizes software agents can be used to build adaptable multicast trees. Initially, the Application Control Agent (ACA) receives the application requirements and then passes these requirements to the Resource Scheduling Agent (RSA) that will determine the appropriate multicast tree. RSA, then, considers the current state of the networks and computing systems by accessing two databases: Communication Cost Database (CCD) and the Machine State Database (MSD).
In the initialization phase of the ACS Multicast Algorithm, an application invokes a group create via *ACS_create_group()*, and *ACS_create_group()* calls first Resource Scheduling Agent(RSA) to build a tree that is contructed from considering both group communication patterns and the computing power as well as network conditions. After an acs tree is returned from RSA, CCA(Connection Control Agent) establishes links between nodes in the tree (See steps 1 through 7 in Figure 4).

Once the application is running using the multicast tree proposed by the RSA, the application agents monitor the performance of ACS multicast services using three procedures (see Figure 5): ACS_Change_Detect, ACS_Verification_Analysis, and ACS_Adaptation_Plan procedures. While the application is running (Step 1 in the ACS Adaptation Algorithm of Figure 5), the ACS control agent monitors all the ACS group services (Steps 2 through 9 in the ACS Adaptation Algorithm of Figure 5). For example, during the multicasting service, the data transmission has experienced an unreachable. This is detected when the ACS monitoring routine discovers that the communication link is fail. Once *link_unavaiable* is recognized, the Verification and Analysis procedure to pinpoint the cause of *link_unavaiable* (Step 6c in the ACS Adaptation Algorithm of Figure 5).

The selection of the appropriate multicast tree for the



RMA(Resource Management Agent), CCD(Communication Cost Db), MSD(Machine Status Db), RSA(Resource Scheduling Agent)
BNF(Best Node First), ACA(Application Control Agent), CCA(Connection Control Agent), AMA(Application Monitor Agent)S
GCA(Group Communication Agent), PPA(Point−to−Point Agent)

Figure 3. Interactions of Adaptive Communication System

```
Application APi(Gp)
    Initialization : /* Gp is a group */
        Call Resource Scheduling Agent(RSA) to build a initial tree
        for Gp
        Call Connection Control Agent(CCA) to setup connections for Gp
    end Initialization
    while (APi) is running do
        ACS_mcast(Gp,m)
    end While
end Application APi

Resource_Scheduling_Agent( APi(Gp))
    Analysis group communication patterns for APi
    Call Application_Aware_Scheduling(APi,Tree)
    Call Resource_Aware_Scheduling(APi,Tree)
end Resource_Scheduling_Agent
```

Figure 4. ACS Group Communication Protocol

ACS group communication services utilizes two algorithms: Application Aware Algorithm(AAA) to adopt a binary tree to support the group communication patterns of an application, and Resource Aware Algorithm(RAA) to assign machines to the nodes of the multicast tree. In what follows, we discuss in further detail these two algorithms.

4. Resource Aware Scheduling(RAS)

In heterogeneous network environment, networks have a wide range of performance characteristics in terms of bandwidth, protocols and topology. In ACS, we take the characteristics of machines and networks when we assign machines to the nodes of a multicast tree. Thus, in this section, we characterize factors leading to heterogeneous computing environments, and how to implement efficient group communication on heterogeneous environments by exploiting one or more of the heterogeneous characteristics.

4.1. Impact of Resource Type

There are several factors that can impact on the performance of group communications in heterogeneous computing environments: communication capability of machines, communication protocol and network architecture. Before

```
procedure ACS_Group_Protocol
    while ($AP_i(G_p)$) is running do /* $G_p$ is a group */
        Set Up Monitoring Environment
        Start check CCD and MSD via Resource Management Agent(RMA)
            EventType ⟵ ACS_Change_Detect(CCD,MSD,$G_p$)
                if EventType ≠ Normal then
                    EventCause ⟵
                    ACS_Verification_Analysis(EventType,$AP_i(G_p)$)
                    ACS_Adaptation_Plan(EventCause,$AP_i(G_p)$)
                endif
    end While
end ACS_Group_Protocol

procedure ACS_Change_Detection(EventType,$AP_i(G_p)$)
    EventType ⟵ Normal
    if communication delay $> D_{thresh_hold}$ then
        EventType ⟵ link_excessive
    else if link down detected then
        EventType ⟵ link_unavaiable
    else if machine load is increased
        EventType ⟵ system_overload
    else if machine is unavailable
        Event Type ⟵ system_unavaiable
    if $MP_k^{n^t}(m) - MP_k^{n^{t-1}}(m) > \Delta_{multicast}$ then
        EventType ⟵ low_performance
    endif
end ACS_Change_Detection

procedure ACS_Verification_Analysis(EventType,$AP_i(G_p)$)
    case EventType = link_excessive
        if | $CCD^t(i,j,bandwidth)$ - $CCD^{t-1}(i,j,bandwidth)$ |>
        $\Delta_{bandwidth}$ for any $i,j \in \{G_p\}, i \neq j$ then
            EventCause = link_excessive
        else if $MSD(I,load) > load_{threshhold}$ then
            EventCause = system_overload
        endif
    case EventType = link_unavaiable
        if $CCD^t(i,j,protocol) =$ unavailable for any $i,j \in \{G_p\}, i \neq j$ then
            EventCause = link_fail
        else if $CCD^t(i,j,hops) = \infty$ for any $i,j \in \{G_p\}, i \neq j$ then
            EventCause = link_fail
        else if $MSD^t(i,down)$ = 'Yes' for any $i \in \{G_p\}$ then
            EventCause = system_down
        endif
        case EventType = lower_performance
        /* Evaluating Multicasting Algorithm */
        if $MP_k^{n^t}(m) - MP_k^{n^{t-1}}(m) > \Delta_{multicast}$ then
            EventCause = decrease_performance
        endif
    Return EventCause
end procedure ACS_Verification_Analysis()

procedure ACS_Adaptation_Plan(EventCause,$AP_i(G_p)$)
    if EventType ∈
    { link_excessive, link_fail, unreachable, powe_decrease, system_down } then
        Call Resource Aware Procedure(RAP) to rebuild a tree for $G_p$
        if EventType = decrease_performance then
                Call Application Aware Procedure(AAP) to rebuild a tree for $G_p$
        endif
        Call Connection Control Agent(CCA) changes connections for $G_p$
    endif
end procedure ACS_Adaptation_Plan()
```

Figure 5. ACS Adaptation Algorithm

we develop an algorithm that takes into consideration of these characteristics, we study the impact of these characteristic on the communications.

We present experimental results to show the effect of communication capabilities of machines on the average round time latency between six different pairs of SUN workstations in a heterogeneous environment. The workstations are connected via Ethernet and used socket based TCP library to communicate. Each workstation sends 8K bytes data to all other workstation. This experimental result is shown Table 1, and shows how the capability of workstation affects the time taken to transmit a message from one workstation to another.

The fastest machine is SUN Ultra 4B and the slowest machine is SUN Sparc Classic. These experimental results show that the average round trip time for a SUN Sparc Classic is around 9 times that of SUN Ultar 4B, and indicates that workstation speeds can have direct impact on the communication round trip time.

4.2. Impact of Network Architecture

In order to study impact of network architecture, we measure the average round trip time between four workstations that are attached on campus LAN environment. We select four workstations that have same configuration (SUN Ultra 5), and one pair of workstations from out of four and their connections are as follows: 1) in the same room, 2) in the same floor but two different rooms, 3) between two different floors, 4) between two buildings.

Each workstation sends 8K, 16K, 32K, 64K bytes data to all other workstation. This experimental result is shown Table 2, and tells how the network topology affects the time taken to transmit a message from one workstation to another.

The shortest distance of the two connections is the same room and the longest distance of the two connections is the different building. These experimental results indicates that the average round trip time for the different building is around 1.26 times that of shortest distance, and demonstrates that the network topology also impact on the communication round trip time.

Group communication operations involve more than one workstations, and thus, if we use the heterogeneity to our advantage to implement the operations can be faster. With these two observation, we introduce a formulae that can represent the communication capacity and network architecture of machines so that we use the formulae to develop a schedule algorithm for ACS multicasting.

	Ultra 4B	Ultra 4	Ultra 60	Ultra 5	Sparc 4	Sparc CL
Ultra 4B	N/A	1.040	8.262	8.652	8.711	9.011
Ultra 4		N/A	8.349	8.544	9.222	9.408
Ultra 60			N/A	9.015	9.499	10.361
Ultra 5				N/A	9.461	10.079
Sparc 4					N/A	10.058

Table 1. Average round trip times in milliseconds between different types of workstations

Size	Same Room	Same Floor	Different Floor	Different Building
8K	7.115	7.194	8.477	9.008
16K	14.345	14.499	17.092	18.143
32K	28.677	28.942	34.113	36.243
64K	57.347	57.915	68.078	71.925

Table 2. Average round trip times in milliseconds between different network topologies

4.3. Communication Power of a Machine

In order to use different characteristics of machines and networks, we need to develop a new metric that can characterize and quantify the computing power of the computers and network resources. A new metric, *Power*, is introduced to quantify and characterize the capacity of computing systems and is then used to build the appropriate Resource Aware Tree (RAT). In general, the power of a machine can be characterized in terms of the CPU speed, size of memory and speed of I/O. However, we ignore the term speed of I/O because we are interested in developing a metric that can impact on the the communication performance.

Equation 1 shows one potential representation of the Power associated with computer$c$. To quantify the power of computers, we denote all terms with same unit. For example, we pick the configuration of SUN SPARC Classic as our reference system that has 50 Mhz CPU and 48 MBytes Memory, which is the slowest machine.

$$\begin{aligned} P(c) &= f(CPU, Main\ Memory) \\ &= W_{cpu}(c) \times N_{cpu}(c) \times CLK(c) + \\ &\quad W_{mem}(c) \times MEM(c) \end{aligned} \tag{1}$$

Where

$N_{cpu}(c)$: Number of processors in machine c

$CLK(c)$: Units of Clock speed = $\lceil \frac{clock\ speed\ in\ Mhz}{50\ Mhz} \rceil$

$MEM(c)$: Units of memory module = $\lceil \frac{size\ of\ memory\ MB}{48\ MB} \rceil$

$W_{cpu}(c)$: CPU Weighted factor

$W_{mem}(c)$: Memory Weighted factor

where $W_{cpu}(c) + W_{mem}(c) = 1.0$

In general, the communication capacity of machine can be changed from time to time. When the loads on a computer increases, the power of the machine should be decreased to reflect the changes in the load. Equation can be modified slightly be dividing the power of the CPU and memory factor by the current load and available memory, respectively, as shown below:

$$\begin{aligned} P^t(c) = W^t_{cpu}(c) \times N^t_{cpu}(c) \times \frac{CLK^t(c)}{L^t(c)} &+ \\ W^t_{mem}(c) \times \frac{MEM^t(c)}{L^t(c)} & \end{aligned} \tag{2}$$

$$Where\ , L^t(c)\ is\ the\ system\ c's\ \ load\ \ at\ t$$

$L^t(c)$ can be obtained by using the monitoring agents that use the appropriate system routines (e.g., uptime, rup in Unix). When there is no process, we assume $L^t(c) = 1.0$. As an example, the current computing power of the machines are stored in the Machine Status Database (MSD) is shown in Figure 6. The power of reference machine is 1.0 and the power of the fastest machine is 24.40 as shown in the example. ACS keeps monitoring the status of system load periodically and updating the *Power* $P^t(c)$ values and store them into the MSD which will be used to select the best available machines for requested multicasting services by resource aware scheduling.

4.4. ACS Resource Aware Scheduling Algorithm

ACS Resource Aware Scheduling algorithm uses network characteristics and computing powers to build the appropriate multicast algorithm. It follows a greedy algorithm in its machine assignment. That is it assigns first the machines with the maximum power as shown in Figure 7. In

Name	Power	#CPU	Speed (Mhz)	MMSize (MB)	Load	MemType	OS	Down
M1	7.20	1	500	128	1.0	L	WIN	No
M2	6.00	1	650	128	1.5	L	Linux	No
M3	4.20	1	500	256	2.0	B	NT	No
M4	14.00	2	366	512	1.0	B	Solaris	Yes
M5	16.00	2	296	1024	1.0	B	AIX	No
M6	1.00	1	50	48	1.0	B	Solaris	Yes
M7	5.40	1	333	128	1.0	B	Solaris	No
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Mp	24.40	2	296	2048	1.0	B	Solaris	No

Figure 6. An example data of MSD

```
    algorithm Resource_Aware_Scheduling _Algorithm
1       Input : LIST(gsize), MSD(Machine Status Database)
2       Output : Resource Aware Tree(RAT);
3           Sort available machines according to its power(using MSD) and
3c          put it into COST list
4           i=0;
5           node = LIST; /* Get the first node from the LIST */
6           while node !=  do
7               /* Assign parent node to the best machine */
7c              RAT(node) = COST(rank(i))
8               /* Assign left child node to the next better machine */
8c              RAT(left(node)) = COST(rank(i+1));
9               /* Assign right child to the 3rd machine. */
9c              RAT(right(node)) = COST(rank(i+2));
10              i=i+3;
11          endwhile
12          end
13          Return RAT to ACA;
    end ACS_Aware_Scheduling_Algorithm

    function rank(i)
1   if i  gsize then
2       /* If group size is bigger than total no. of available machines then
2c      i=0; 3  Assign a machine to a node from the best machine */
4   endif
5       return (i);
    end rank
```

Figure 7. ACS_Aware_Scheduling _Algorithm

this algorithm, we first sort the machines according to their power (step 3). Once the machines are sorted, we assign the machines to the tree nodes by traversing from the root to the leaf nodes following breath-first traversal strategy (steps 5-10). Thus, the master node is always assigned to the fastest machine and second level nodes are assigned to next fastest machines from left nodes to right nodes. By this algorithm, leaf nodes are the slowest machines or the same speed machine as their parent nodes.

Figure **??** shows the benefit of a resource aware scheduling algorithm when compared with an assignment that does not take the power of the machines into consideration. It is also clear the performance gap will increase as the message size increases.

When RSA assigns machines to the nodes of a multicasting tree or estimates the performance of multicasting algorithm, it utilizes a Communication Cost Database(CCD) that contains estimated measurements of network bandwidth between different machines, available protocols and network technologies. The CCD consists of average bandwidth, type of communications protocol, number of hops. The C(i,j,bandwidth) denotes the average time to send the a message from machine M_i to M_j. The C(i,j,protocol) denotes the communication protocol to be used to transmit a message from machine M_i to M_j. The C(i,j,hops) denotes the number of hops to exchange a message between machine M_i and M_j. Figure 10 shows an example CCD table.

The ACS Multicasting Algorithm utilizes the CCD to assign resources to the multicast tree nodes. The CCD is kept up-to-date by the Resource Management Agent(RMA). The RMA periodically collects and saves the latest network status and information using SNMP network agents and/or special monitoring agents. The RMA runs a network benchmarking tools such as *ping, netperf* to

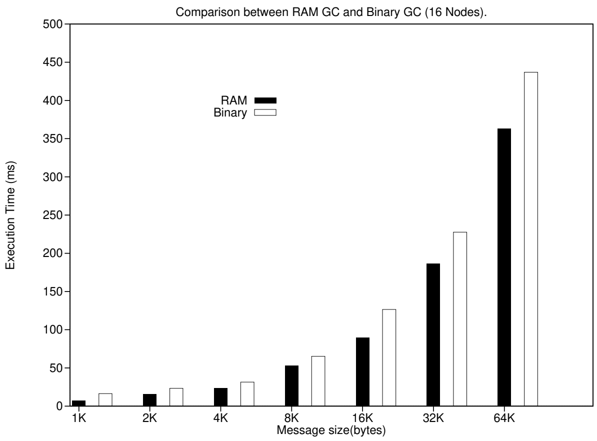


Figure 8. Multicasting Performance using Resource Aware Multicasting(RAM) with 16 nodes

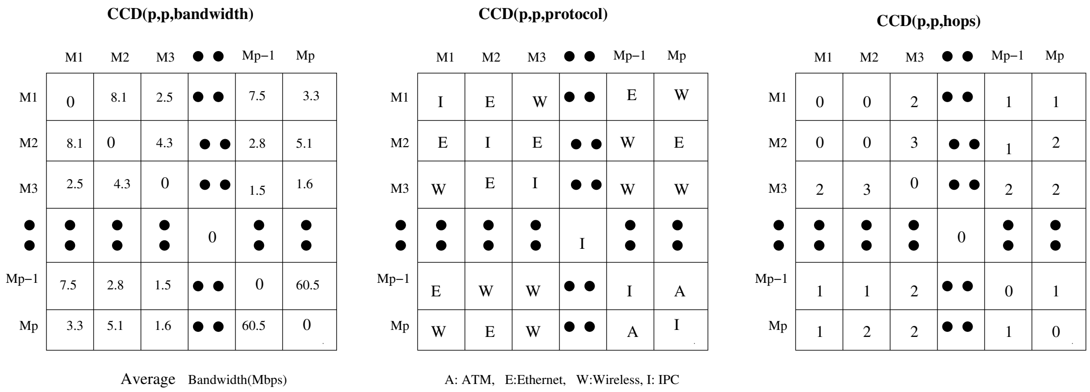


Figure 10. An example data of CCD

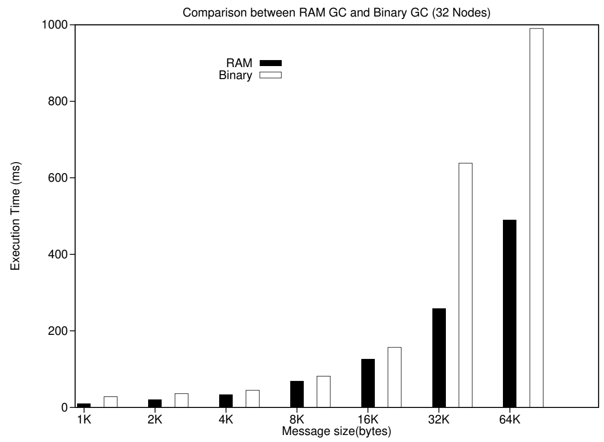


Figure 9. Multicasting Performance using Resource Aware Multicasting(RAM) with 32 nodes

collect performance metrics when the network condition is changed due to fault or overload.

The basic premise of the Resource Aware Scheduling(RAS) is that the performance of a binary tree can be improved if we take into consideration the machine type and performance when we assign tree nodes to computers. For example, let us assume that we need to broadcast a message to a group consists of 16 nodes. We have studied three assignments of resources to the binary tree as outlined in Resource Aware Tree 1 (RAT1), RAT2, and RAT3. The characteristics of the machines and networks assigned to these trees are summarized is Table 3.

The ACS multicast may choose any one of these three different resource scheduling trees to execute multicasting requets. This can be done by calling the ACS_mcast() as shown below:

1. ACS_Group_Create(**RAT1**,....); ....
   ACC_mcast(**RAT1**,16,int,10,msgbuf,{1K,..,64K})

2. ACS_Group_Create(**RAT2**,....); ....
   ACS_mcast(**RAT2**,16,int,10,msgbuf,{1K,..,64K})

3. ACS_Group_Create(**RAT3**,....); ....
   ACS_mcast(**RAT3**,16,int,10,msgbuf,{1K,..,64K})

Figure 11 shows the response time of the application using three different resource allocations. From this experiment, it is clear that the clock speed of a processor has more impact on the multicast performance. The response time for an application using the three allocations (RAT1, RAT2, and RAT3) is illustrated in Figure 11.

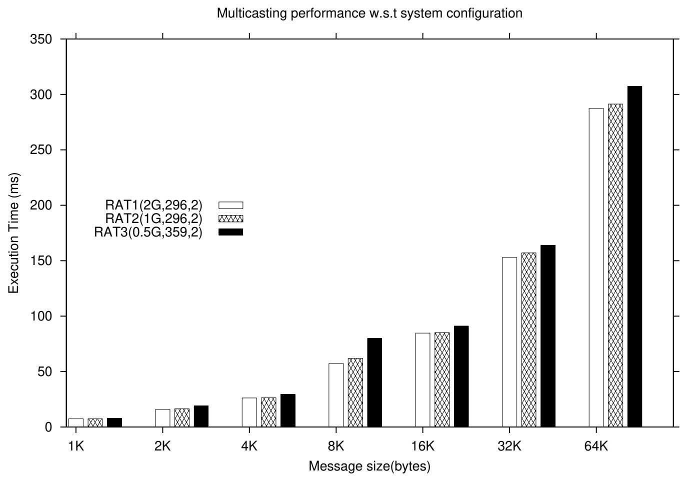


Figure 11. Application execution time for three resource allocations

Scheduling	master node	other nodes (15 nodes)
RAT1	2 cpus, 296 MHz, 2GB	1 cpu, 333 Mhz, 128 MB
RAT2	2 cpus, 296 MHz, 1GB	1 cpu, 333 Mhz, 128 MB
RAT3	2 cpus, 359 MHz, 0.5GB	1 cpu, 333 Mhz, 128 MB

Table 3. An example of Resource Aware Scheduling

5. Application Aware Scheduling

As number of nodes are increased in the binary tree, its performance is getting worse because it send more message and therefore incur more message startup overhead. Therefore, we can improve the performance of multicasting by changing a binary multicasting tree to application aware tree that is build based on group communication pattern. In this Section, we present how we can build an application aware tree from a binary tree using analytic approach, and also show the performance improvement of the multicasting by AAM.
In this Section, we present how we can build an application aware multicasting tree from a binary tree using an analytical model.

5.1. Definitions

We are capturing applications performance using analytical techniques and measuring. Before we elaborate the each time values, we make some assumptions for sake of simplicity. We assume a given tree is fully balanced tree. Therefore, the total number of links from the root node to a leaf node is always same in the given tree. Let us define the following information :

- *k*= degree of a given multicasting tree
- *h*= the number of links from the root node to a leaf node
- *n*= size of a group
- *N*= total requested number of multicasting with in a group
- *t(m)*= average data transfer time for message size *m*
- *o(m)*= average overhead time for writing message size *m*
- *a(c)*=average delay time for acknowledge size *c*

We introduce a function $MP_k(m)$, called *Multicasting Performance Function(MPF)*. The MPF for ACS group communication consists of four different time components, and can be presented as follows:

$$MP_k^n(m) = T_k^n(m) + D_k^n(m) + A_k^n(c) + C_k^n(m) \quad (3)$$

Each term can be defined as follows:

Data Transfer time($T_k^n(m)$)

Data transfer time for *m* bytes data through multicasting *k*-ary tree with n nodes. In multicasting environment, data transfer time is determined the longest transfer time, the heights of the given tree, because transferring data is done in parallell manner. The data transfer time from the root node to a leaf node *i* can be defined in Equation 4.

$$\begin{aligned} t_k^{n^i}(m) &= h_i \times t(m), \quad (4)\\ where, &\quad h_i = \lceil \log_k n_i \rceil - 1 \\ &\quad n_i \in \{n_1, n_2, ..., n_n\} = n \end{aligned}$$

The multicasting time for a given tree, in general, can be define as the one of the longest transfer time among data transfer times from the master group server(root node) to a leaf node because a transferring data performs parallelly after a message transfered to local group servers. Thus, the multicasting time for a give tree will be the longest transfer time that is dependent on *h* which is the length of path from the root to a leaf node. Since the *h* is the same value for any leaf node in a balanced tree, $t_k^n(m)$ can be defined as Equation 5.

$$\begin{aligned} t_k^n(m) &= \max(t_k^{n^1}(m), ..., t_k^{n^n}(m)) \\ &= t_k^{n^i}(m), where\ i \in 1, .., n \quad (5) \end{aligned}$$

With Equations 4 and 5 we can now define the function of multicasting time for a given *k*-ary tree,$T_k^n(m)$ as in Equation 6.

$$T_k^n(m) = t_k^n(m) = h \times t(m), where\ h = \max(h_1, h_2, .., h_n) \quad (6)$$

Now, we can calculate the total multicasting time for requesting N-times multicasting in a *k*-ary tree by multiplying $T_k^n(m)$ by N, i.e., $T_k^n(m) \times N$.

Receiving Delay time($D_k^n(m)$)

In a multicasting tree, there is a delay time due to waiting for receiving the first byte of *m* byte(s) data from their master(root) node in a *k*-ary tree with n nodes. It depends on the location of a child node in a tree. In ACS group communication, the sending messages begins from left to right and top to bottom. For example, k-*th* child node has to wait for (k-1) delay times in order to receive data from its parent node. The receiving delay for leaf node l, $d_k(l)$, can be built as in Equation 7.

$$\begin{aligned} d_k(l) &= \sum_{i=1}^{h} mod(p_i, k) \qquad (7) \\ where \quad & h = heights\ of\ a\ tree = \lceil \log_k n \rceil - 1 \\ & p_i = \lfloor \frac{p_{i-1}}{k} \rfloor - 1, where\ i > 1 \\ & p_1 = l - 1 \end{aligned}$$

Like the data transfer time for multicasting, the delay time for a multicasting can resent the longest delay among leaf nodes. Thus, the delay time for n nodes multicasting tree,$w_k(n)$, can be described as follows in Equation 8.

$$\begin{aligned} d_k(n) &= \max(d_k(l_1), ..., d_k(l_n)) \\ &= d_k(l_n). \end{aligned} \qquad (8)$$

Thus, the delay time for receiving *m* bytes data from root node in a tree can be obtained using Equation 9, and it follows:

$$D_k^n(m) = d_k(m) \times o(m) \qquad (9)$$

If there are N times multicasting requests in a given group, the delay time for this, $D_k^n(m) \times N$.

Acknowledgment Delay time($A_k^n(c)$)

ACS group communication supports reliable group communication as a default. When a group master sends a message to its group members, it waits for acknowledgment from all members. If the acknowledgment is not received by parents within timeout period, the master tries to retransmit the message to members. Although the size of acknowledge message size is relatively small, the waiting time for acknowledgment affects the performance of multicasting algorithm. For sake of simplicity, we assume there is no retransmission by application level if we use the TCP protocol stack. The following Equation in 10 is based on the above assumption. Acknowledgment is an answer back for transmitted message from a parent node, and the total number of acknowledgment is the same as the total number of data transfer requested. Thus, the acknowledgment delay time could be found as the same as finding data transfer time. Using the same formulae in Equations 4 and 5, we get the following equation which is almost same as the Equation 6.

$$A_k^n(c) = h \times a(c), where\ \ h\ = \lceil log_k n \rceil - 1 \qquad (10)$$

Network Congestion Time($C_k^n(m)$)

Although we count the transfer time, receiving delay as well as acknowledge delay for estimating multicasting time, there are still some other delays which we can't simply either measure or formulate. In a multicasting tree, many of machines are concurrently involving sending and receiving data as well as acknowledgment, and these are impact on the performance. We define such a delay as network congestion delay time that includes delay time due to retransmission message by network, delay time due to network contention if ACS multicasting uses CSMA/CD, and system initial overhead before sending a message. The common characteristic of these times is that these delay times are dependent on the size of requested message.

To find an analytical model to represent the network congestion delay, we analyze the measured multicasting performance data and try to find a relationship between the performance data and message size. With none-linear regression modeling method, we introduce the a formula which represents the network congestion delay with respect to the message size, and the formula is shown in Equation 11

$$f_k(m) = \frac{e^{\beta}}{m^{\alpha}} \qquad (11)$$

Since there are too big gaps for the size of data between 1 byte to 72K bytes, we divide several intervals and find the best α and β for different type of trees. The $f_k(m)$ can be defined in Table 4.

5.2. ACS Application Aware Scheduling Algorithm

In this subsection, we develop collective communication techniques that enable applications to adapt to group communication patterns. We assume we know size and pattern of group communications before we make a communication schedule. We use a communication model to analytically represent that has been discussed in the previous subsection and use the Equation 3. When ACS group communication services start, ACS control and management agent adds application_aware_scheduling_agent() to build an Application Ware Tree (AWT) that adapts application communication patterns. It invokes algorithm 12. First ACS builds a binary tree with group size *n*(Step 2 in Algorithm 12). Second, ACS also constructs a two level group communication tree based on group communication patterns with group size

Mesg Size	Binary		Adaptive	
Byte(s)	α	β	α	β
1,8,16	0.017684	4.895701	0.046907	4.253952
32,64,128	0.252116	4.205185	0.264265	3.609830
256,512,1K	0.801424	1.399798	0.101838	5.628978
2K,4K,8K	0.458562	4.057544	0.721115	1.719207
16K,32K,64K,72K	0.487319	4.099067	0.145196	7.478510

Table 4. constant for a waiting time function

n(Step 3 in Algorithm 12). Compute estimated communication time for built trees, and compare their values(Step 4 and 5, in Algorithm 12). Select the low cost value one with the results (Step 6 to 9 in Algorithm 12).

Application_Aware_Scheduling_Agent($AP_i(G_p)$)
1 Analysis group communication patterns for AP_i
2 Build a binary tree for AP_i
3 Build a 2 level tree with the result of analysis
4 Evaluate each tree with $MP_k(m)$, CCD and MSD
5 **if** $MP_{2L}(m) < MP_2(m)$ **then**
6 Tree $\longleftarrow$ 2 Level Tree
7 **else**
8 Tree $\longleftarrow$ Binary Tree
9 **endif**
end Application_Aware_Scheduling_Agent

Figure 12. ACS Application_Aware_Scheduling_Agent()

Multicasting Application:
/* Assume $G0 = \{m_1, m_2, ..., m_{16}\}$,
$G1 = \{m_1, m_2, .., m_8\}, G2 = \{m_9, m_{10}, m_{11}, m_{12}\}$,
$G3 = \{m_{15}, m_{16}\}$
Thus, $G1, G2, G3 \subset G0$
$|G0| = 16, |G1| = 8, |G2| = 4, |G3| = 2$
msgsize = { 1,8,16,32,64,128,256,512,1K,2K,4K,8K,16K,32K,64K,72K } */
.....
for i= 1 to 10
/* multicasts a message 10 times to group G0 */
ACS_mcast(G0,int,10,msgbuf,msgsize);
endfor
...
for j= 1 to 25
/* multicasts a message 25 times to group G1 */
ACS_mcast(G1,int,10,msgbuf,msgsize);
endfor
...
for k= 1 to 50
/* multicasts a message 50 times to group G2 /*
ACS_mcast(G2,int,10,msgbuf,msgsize);
endfor
...
for l= 1 to 100
/* multicasts a message 100 times to group G3 */
ACS_mcast(G3,int,10,msgbuf,msgsize);
endfor
...
End Multicasting Application

Figure 13. An application example for using ACS multicasting

6 Illustrative Examples

Assume there is an application that has one group and 3 subgroups for multicasting requests, and its actual implementation in ACS is as follows: We evaluate the performance of this application with respect to two multicasting trees: (1) Binary Tree Algorithm, and (2) 2-Level Tree. These two multicasting trees are constructed when the *ACS_group_create()* is called as shown below:

- Binary Tree :
  ACS_group_create(G0,TCP,none,err1,**Binary**,QoS)
- 2-Level Tree :
  ACS_group_create(G0,TCP,none,err1,**2-Level**,QoS)

The corresponding trees for these two calls are shown in Figure **??** .

We run two different types of trees with an application which requests sending messages using multicasting communication service in order to achieve two goals:

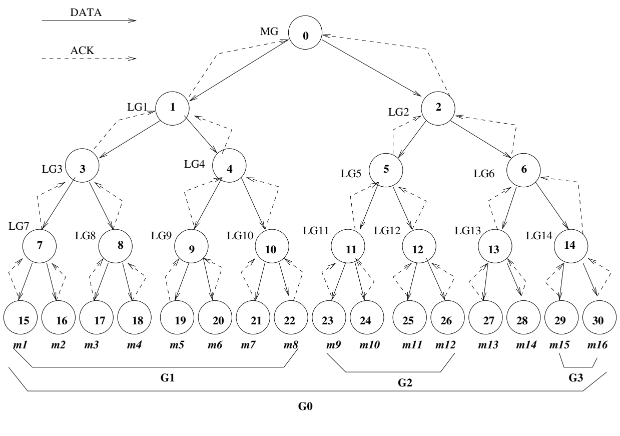


Figure 14. Binary-Tree Configuration

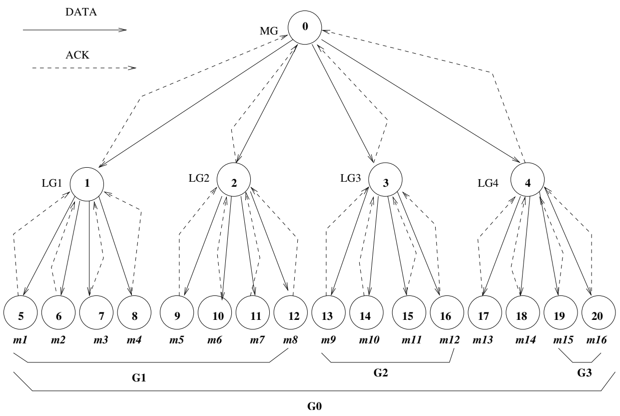


Figure 15. 2-LevelTree Configuration

- To see the performance difference between a binary tree and other tree
- To check the effectiveness of the multicasting performance function that we derived from the previous section.

In this example we compare the performance of the ACS multicast algorithm with o ur analytical performance function $MP_k(m)$ for different message sizes. The mu lticast performance can be described in terms of the delays related to data tran sfer, message processing delays at each receiver node, and the delay related to acknowledgment for each group (G0, G1, G2, and G3) as shown in Table 5.

To predict the application multicast performance for the two trees, we determine experimentally three functions to estimate the delays due to network traffic $f_k(m)$, the processing delays at each computer $t(m), o(m), a(4B)$. The values of these functions are shown in Table 6.

The application multicast performance for the binary tree and 2-level tree can be predicted using the following two Equations 12:

$$\begin{aligned} MP_2(m) &= 740 \times t(m) + 315 \times o(m) + \\ &\quad 740 \times a(c) + f_2(m) \\ MP_{2L}(m) &= 370 \times t(m) + 410 \times o(m) + \\ &\quad 370 \times a(c) + f_{2L}(m) \end{aligned} \tag{12}$$

The application performance for both measurement and the our prediction function is in Table 7. The error between predicted multicasting time and measured multicasting is within 0.2 % to 3.3 % range. The multicast function c an be used to predict the performance of the multicast algorithm and can be used to identify the appropriate multicast tree structure suitable for any given app lication.

We have also compared the performance of ACS with other message-passing tools by measuring the execution time of a static voting application that requires intensive group communications.

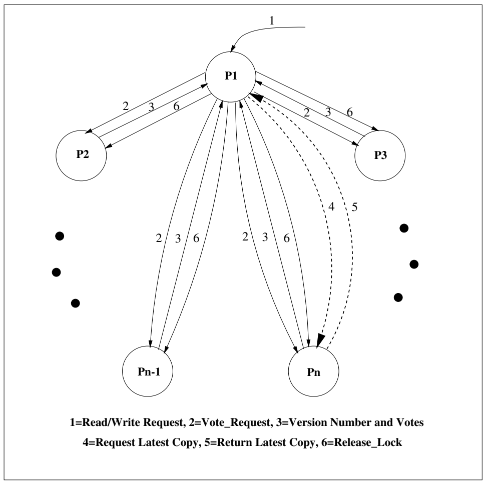


Figure 16. Static Voting Algorithm

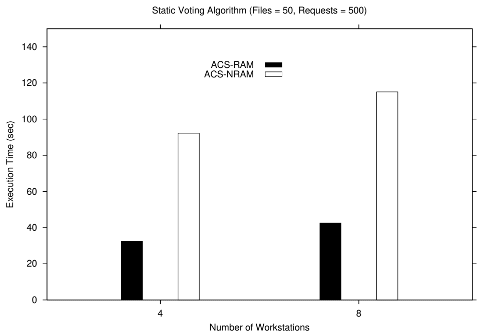


Figure 17. Comparison of Application Performance

Replicating data at different locations is a common approach to achieve fault tolerance in distributed computing systems. One well-known technique to manage replicated data is voting mechanisms. The algorithm used in this experiment is based on the static voting scheme proposed by Gifford [16]. In this algorithm (See Figure 16) we assume that there is a file server process in each node that handles *read* and *write* requests for a given file. Each file server process issues arbitrary *read* and *write* requests that were produced randomly using a random number generator. Whenever a server process issues a file access request, it sends a *Lock_Request* message for that file to the local lock manager and broadcasts a *Vote_Request* message to all other server processes. When the server process receives a *Vote_Request* message from other server processes, it sends a *Lock_Request* message for the requested file to

Delay Time		Group				Total
Time	Tree	G0	G1	G2	G3	Time
Data Transfer	$T_2^{x^i}(m)$	$4*t(m)*10$	$4*t(m)*25$	$4*t(m)*50$	$4*t(m)*100$	$740*t(m)$
Data Transfer	$T_2^{x^i}(m)$	$4*t(m)*10$	$4*t(m)*25$	$4*t(m)*50$	$4*t(m)*100$	$740*t(m)$
	$T_{2L}^{x^i}(m)$	$2*t(m)*10$	$2*t(m)*25$	$2*t(m)*50$	$2*t(m)*100$	$370*t(m)$
Receiving	$D_2^i(m)$	$4*o(m)*10$	$3*o(m)*25$	$2*o(m)*50$	$1*o(m)*100$	$315*o(m)$
	$D_{2L}^i(m)$	$6*o(m)*10$	$4*o(m)*25$	$3*o(m)*50$	$1*o(m)*100$	$410*o(m)$
Acknowledge	$A_2^i(c)$	$4*a(c)*10$	$4*a(c)*25$	$4*a(c)*50$	$4*a(c)*100$	$740*a(c)$
	$A_{2L}^i(c)$	$2*a(c)*10$	$2*a(c)*25$	$2*a(c)*50$	$2*a(c)*100$	$370*a(c)$

Table 5. Group multicast component delays

Times	1B	64B	1KB	64KB
$t(m)$	0.184 ms	0.238 ms	1.033 ms	57.216 ms
$o(m)$	0.036 ms	0.037 ms	0.040 ms	45.937 ms
$f_2(m)$	133.7137 ms	191.2738 ms	1048.2170 ms	13408.0111 ms
$f_{2L}(m)$	70.3830 ms	110.9276 ms	563.8934 ms	8855.4217 ms
$a(4B) = \ 0.185\, ms$, where acknowledgment is 4 Bytes only				

Table 6. Experimental functions to predict application multicast performance

Mesg Size	Adaptive			Binary		
Byte(s)	Measured(ms)	Predict(ms)	Error(%)	Measured(ms)	Predict(ms)	Error(%)
1	225.434	221.673	1.668	420.610	418.114	0.594
64	276.409	282.608	2.243	504.950	515.949	2.178
1K	1066.281	1030.952	3.313	1965.269	1962.135	0.159
64K	49023.240	48927.978	0.194	70447.749	70354.916	0.132

Table 7. Application Multicast Performance using measurement and analytical techniques

the local lock manager. The server process then returns the version number of the replica and the number of votes assigned to the replica to the server process that initiated the *Vote_Request*. Based on the information returned from other server processes, the server process decides if the file access is granted and the file is the latest copy. If the local copy is different from those replicated at other server processes, it test copy from other server processes. Finally, the file server process broadcasts a *Release_Lock* message to all other file servers if the file access is granted. In this experiment we assumed that there are 50 different files replicated at each node and each file server process generates 500 *read* or *write* requests for arbitrary files. As we present in Figure 17, in static voting application where the size of the broadcasting messages are small and the communications take place randomly, the performance of Resource Aware Multicasting ACS is comparable to that of Non-Resource Aware Multicasting ACS for small size groups but the performance gap gets wider as we increase the group size. We believe that most of the improvements of Resource Aware ACS are due to consideration of heterogeneity of computing resource.

7 Conclusions

In this paper, we have presented the ACS architecture and its group communication n services. We have also developed novel analytical techniques to predict the pe rformance of application multicast algorithms and proposed to used to build appl ication-centric multicast algorithms. We also demonstrated how we can build an application aware multicasting algorithm from a binary tree algorithm to improve its performance. The experimental results of using the ACS multicasting algorithm showed that the predicted time is closed to the measured time up to 0.2% , and the performance of multicasting by application-aware tree gives better performance than of the algorithm based on a binary tree.

References

[1] M. Banikazemi, V. Moorthy, and D. K. Panda, Efficient collective communication on heterogeneous networks of workstations. *In Proc. Intl. Conf. Parallel Processing*, pp. 460-467, 1998.

[2] P. B. Bhat, C.S. Ragphavendra, V. K. Prasanna Efficient collective communication in Distributed Heterogeneous Systems *In Proc. IEEE Intl. Conf. Distributed Computing Systems*, June 1999.

[3] R. Butler and E. Lusk. Monitors, Message, and Clusters: The p4 Parallel Programming System. *Parallel Computing*, Vol. 20, pp. 547–564, April 1994.

[4] P. Druschel, L. L. Peterson, and B. S. Davie. Experiences with a High-Speed Network Adaptor: A Software Perspective. *Proc. of SIGCOMM*, 1994.

[5] T. Eicken, A. Basu, V. Buch, and W. Vogels. U-Net: A User-Level Network Interface for Parallel and Distributed Computing. *Proc. of the 15th ACM Symposium on Operating Systems Principle*, December 1995.

[6] I. Foster, C. Kesselman, and S. Tuecke. The Nexus Approach to Integrating Multithreading and Communication. *Journal of Parallel and Distributed Computing*, 1996.

[7] I. Foster and C. Kesselman, Globus: a metacomputing infrastructure toolkit, in *Proceedings of the Workshop on Environment and Tools for Parallel Scientific Computing*, 1996.

[8] A. Grimshaw and W. Wulf, Legion - A View from 50,000 Feet, *Proceedings of Fifth IEEE International Symposium on High Performance Distributed Computing*, 1996, pp. 89–99.

[9] B. Gropp, R. Lusk, T. Skjellum, and N. Doss. Portable MPI Model Implementation. Argonne National Laboratory, July 1994.

[10] M. Haines, D. Cronk, and P. Mehrotra. On the design of Chant: A Talking Threads Package. *Proc. of Supercomputing '94*, pp. 350–359, November 1994.

[11] N. Moden, D. Cohen, R. Felderman, A. Kulawik, C. Seitz, J. Seizovic, and W. Su. Myrinet: A Gigabit-per-second Local Area Network. *IEEE Micro*, Vol. 15, No. 1, pp. 29–36, February 1995.

[12] S. Park and S. Hariri. A High Performance Message Passing System for Network of Workstations. *The Journal of Supercomputing*, Vol. 11, pp. 159–179, 1997.

[13] D. C. Schmit. The Adaptive Communication Environment. *Proc. of 11th and 12th Sun User Group conference*, San Francisco, California, June 1993.

[14] L. Smarr and C. Catlett, Metacomputing, Communications of the ACM, 35, 6, (June 1992) 44–52.

[15] V. Sunderam. PVM: A Framework for Parallel Distributed Computing. *Concurrency: Practice and Experience*, Vol. 2, No. 4, pp. 315–340, December 1990.

[16] D. K. Gifford, "Weighed Voting for Replicated Data", *Proc. of the 7th ACM Symposium on Operating System*, pp. 150–162, December, 1979.

[17] P. Mckinley, H. Xu, A. Esfahanian, and L. Ni, Unicasting-based Multicast Communication in Wormhole-routed Networks. *IEEE Trans. on Parallel and Distributed Systems*, Vol. 5, No. 12., pp. 1252–1265, December 1994.