



Symmetric active/active metadata service for high availability parallel file systems

Xubin He^{a,*}, Li Ou^b, Christian Engelmann^c, Xin Chen^a, Stephen L. Scott^c

^a Department of Electrical and Computer Engineering, Tennessee Technological University, United States

^b Scalable Systems Group, Dell Inc., United States

^c Computer Science and Mathematics Division, Oak Ridge National Laboratory, United States

ARTICLE INFO

Article history:

Received 5 February 2008

Received in revised form

16 March 2009

Accepted 18 August 2009

Available online 31 August 2009

Keywords:

Metadata management

Fault tolerance

High availability

Parallel file systems

Group communication

ABSTRACT

High availability data storage systems are critical for many applications as research and business become more data driven. Since metadata management is essential to system availability, multiple metadata services are used to improve the availability of distributed storage systems. Past research has focused on the active/standby model, where each active service has at least one redundant idle backup. However, interruption of service and even some loss of service state may occur during a fail-over depending on the replication technique used. In addition, the replication overhead for multiple metadata services can be very high. The research in this paper targets the symmetric active/active replication model, which uses multiple redundant service nodes running in virtual synchrony. In this model, service node failures do not cause a fail-over to a backup and there is no disruption of service or loss of service state. A fast delivery protocol is further discussed to reduce the latency of the total order broadcast needed. The prototype implementation shows that metadata service high availability can be achieved with an acceptable performance trade-off using the symmetric active/active metadata service solution.

© 2009 Elsevier Inc. All rights reserved.

1. Introduction

High availability data storage systems are critical for many applications as research and business become more data driven. A file system typically consists of two types of data: user data and metadata. Metadata is essential to system availability, because it defines how a file system utilizes its storage space to manage user data. Since metadata is “the data of the data”, disruption of metadata access could result in a failure of the entire I/O system, while the loss of user data normally only affects some user files. Any I/O request can be classified into either a user data or metadata request.

In a traditional storage system [31,11,42], metadata is stored and managed by dedicated metadata services. There are three major components in such a typical storage system: metadata services, data services, and clients. A metadata service maintains information about files and directories in a file system. Data services store file data. Clients send requests to the metadata service and data services to store and retrieve file data. This system architecture has been proved to be very efficient. However, it also implies

several reliability deficiencies resulting in system-wide availability and serviceability issues [19]. An entire distributed storage system depends on the metadata service to function properly. It is a single point of failure.

One way to improve the availability of parallel file systems is to deploy multiple metadata services. Multiple services back each other up. As long as at least one metadata service is alive, the entire system does not fail. Several models exist to perform reliable and consistent replication of the service state to multiple redundant services for high availability. Past research has focused on the active/standby model [2,53,22], where each service has at least one redundant idle backup. However, interruption of service and even loss of service state may occur during a fail-over depending on the replication technique, such as hot, warm or cold standby.

The research presented in this paper targets the symmetric active/active replication model [20] for metadata service high availability, which uses multiple redundant service nodes running in virtual synchrony [32]. In this model, service failures do not cause a fail-over to a backup and there is no disruption of service or loss of service state. All services are active and ready to serve requests from clients. This architecture improves the availability and reliability.

Total order communication [7,18] is important for the symmetric active/active replication model, but the agreement on the total order usually bears a cost of performance. A *fast delivery protocol* is discussed to reduce the latency of totally ordered broadcasting. The

* Corresponding address: Department of Electrical and Computer Engineering, Tennessee Technological University, Box 5004, 38505, Cookeville, TN, United States.
E-mail address: hexb@tntech.edu (X. He).

protocol performs well for both idle and active services. The results show that, for write requests, the performance degradation is acceptable for typical distributed storage systems, and the throughput of read requests increases linearly with the number of services. We are able to show that metadata service high availability can be achieved without interruption of service and with an acceptable performance trade-off using the symmetric active/active metadata service solution.

This paper is organized as follows. The symmetric active/active replication model is discussed in Section 2. Section 3 defines the system model. The services of a total order broadcasting system is discussed in Section 4. Section 5 illustrates the *fast delivery* total order communication protocol. Section 6 describes the symmetric active/active metadata service design in more detail. Experimental results are presented in Section 7. The related work is examined in Section 8, and conclusions are drawn in Section 9.

2. Symmetric active/active replication

The symmetric active/active replication model [20] allows one to provide high availability for any type of client/server scenario using the well-known state machine replication concept [27,43] that relies on a group communication system [13,15] for totally ordered and reliably delivered messages in a virtual synchronous service group [32].

The symmetric active/active metadata service allows more than one redundant service to be active, i.e., to accept state changes, while it does not waste system resources, as seen in an active/standby model. Furthermore, there is no interruption of service and no loss of state, since active services run in virtual synchrony without the need to fail over. The size of the active service group is variable at runtime; i.e., services may join, leave, or fail. Its membership is maintained by the group communication system in a fault tolerant, adaptive fashion, ensuring group messaging properties. As long as one active service is alive, the state is never lost, state changes can be performed, and output is produced according to state changes.

The concept of internal symmetric active/active replication (Fig. 1) allows each active service of a replicated service group to accept query and request messages from external clients individually, while using a process group communication system for total state change message order and reliable state change message delivery to all members of the service group. All state changes are performed in the same order at all services; thus virtual synchrony is given. Consistently produced service group output may be routed through the process group communication system for at-most-once delivery if dependent clients, services, and users cannot handle duplicated messages.

For example, a networked server that changes its state based on remote procedure calls (RPCs), such as the metadata service of a parallel file system, is modified to replicate all state changes in the form of messages to all services in the service group. Upon delivery, state changes are performed in virtual synchrony. The RPCs and respective state changes are decoupled and executed by separate event handler routines. RPC return messages may be unified via the process group communication system, delivered by every process group member, or delivered by only one process group member and temporarily cached by others.

The developed proof of concept prototype is a customized implementation for offering symmetric active/active high availability for the metadata service. It is based on the internal RPC and state change mechanisms of the original metadata service implementation and utilizes adaptors as part of the internal replication architecture (Fig. 1) to provide symmetric active/active high availability without any interruption of service and without any loss of state.

3. Model and definition

Symmetric active/active metadata services for a file system do not allow group partitions, otherwise a single global state for the file system cannot be maintained. There is only one primary group providing metadata services to clients. Any service that leaves the primary group because of either malfunction or network partition should stop responding to any requests from clients.

Assume that the primary group P consists of a group of metadata services $\{p_1, p_2, \dots, p_n\}$. We define S_p^t as the state of metadata in a service p at a time t . We use the notation G^t if the metadata in the group P is consistent at time t .

Property 1. *The metadata in the primary group P is consistent at time t if and only if any two services in P have the same state at time t .*

$$G^t \iff \forall p, q \in P \quad (S_p^t = S_q^t).$$

We define an initial state S_p^0 for a service p in the primary group P , and we assume that $\forall p, q \in P, (S_p^0 = S_q^0)$. m_p^i is the i th message which changes the state of a service p since the initial state S_p^0 . The function $time(m_p^i)$ returns the time a message m_p^i arrives at the service p . $S_{p,i}$ is the state of service p after the message m_p^i .

\mathcal{Q}_p^i represents a sequence of ordered messages sent to service p since the initial state S_p^0 , and the last message of the sequence is m_p^i . $\{\mathcal{Q}_p^i\}$ defines the message set of the sequence: $\{\mathcal{Q}_p^i\} = \{m_p^1, m_p^2, \dots, m_p^i\}$. $|\mathcal{Q}_p^i|$ is the number of messages in the sequence. $\vec{\mathcal{Q}}_p^i$ defines the message order of the sequence:

$$\forall m_p^j, m_p^k \in \{\mathcal{Q}_p^i\}, j < k \implies (time(m_p^j) < time(m_p^k)).$$

Property 2. *A message sequence \mathcal{Q}_p^i is said to be the same as a message sequence \mathcal{Q}_q^j if and only if the message set and message order of the two sequences are the same.*

$$\mathcal{Q}_p^i = \mathcal{Q}_q^j \iff (\{\mathcal{Q}_p^i\} = \{\mathcal{Q}_q^j\}) \wedge (\vec{\mathcal{Q}}_p^i = \vec{\mathcal{Q}}_q^j).$$

A sequence of ordered messages sent to service p changes the state of a service p from the initial state S_p^0 to $S_{p,i}$. We define a recurrence relation and a function \bar{T} such that

$$S_{p,i} = \bar{T}(S_{p,i-1}, m_p^i)$$

$$S_{p,0} = S_p^0.$$

According to the definition of \mathcal{Q}_p^i , a sequence of ordered messages sent to service p since the initial state S_p^0 , the recurrence relation can be solved to obtain a non-recursive definition of a state transformation function T .

$$S_{p,i} = T(S_p^0, \mathcal{Q}_p^i).$$

Property 3. *If the message sequence \mathcal{Q}_p^i is the same as the message sequence \mathcal{Q}_q^j , then the state of service p , $S_{p,i}$, is said to be the same as the state of service q , $S_{q,j}$.*

$$\mathcal{Q}_p^i = \mathcal{Q}_q^j \implies S_{p,i} = S_{q,j}.$$

Proof.

$$S_{p,i} = T(S_p^0, \mathcal{Q}_p^i), \quad S_{q,j} = T(S_q^0, \mathcal{Q}_q^j),$$

$$\text{since } S_p^0 = S_q^0, \quad \text{if } \mathcal{Q}_p^i = \mathcal{Q}_q^j,$$

$$\text{then } S_{p,i} = S_{q,j}. \quad \square$$

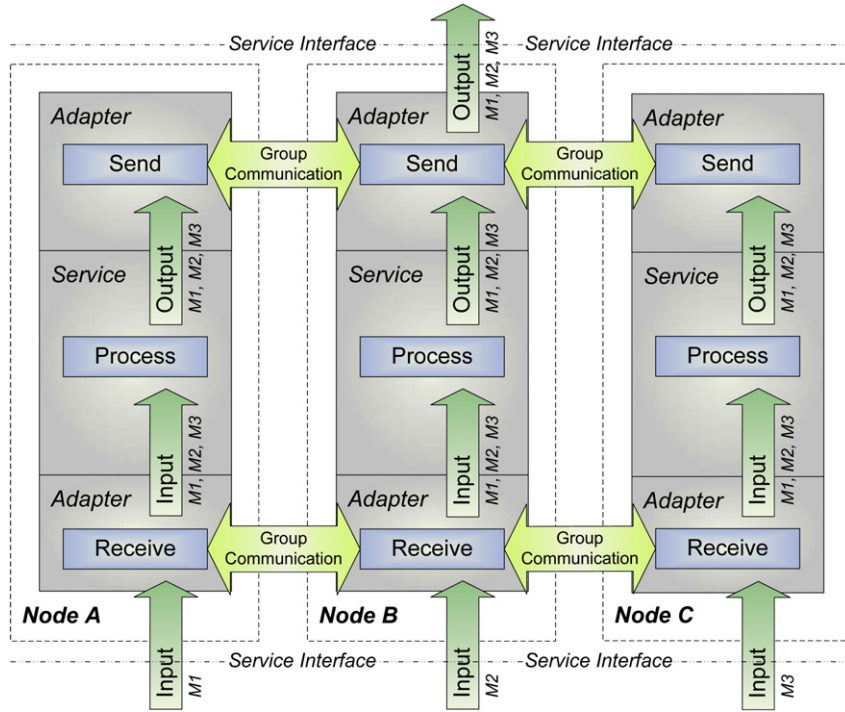


Fig. 1. Symmetric active/active replication architecture using internal replication by service modification/adaptation. The service interface could be a metadata service interface if this model is used to develop active/active high availability for the metadata service.

The function $time(Q_p^i)$ returns the time the last message of sequence Q_p^i arrives at the service p . \bar{Q}_p^t defines the longest message sequence for service p before time t .

$$\bar{Q}_p^t = Q_p^i \rightarrow \bar{A}Q_p^j, \quad (|Q_p^j| > |Q_p^i|) \wedge (time(Q_p^i) < t) \wedge (time(Q_p^j) < t).$$

Property 4. If the message sequence \bar{Q}_p^t is the same as the message sequence \bar{Q}_q^t , the state of service p , S_p^t , is said to be the same as the state of service q , S_q^t , at a time t .

$$\bar{Q}_p^t = \bar{Q}_q^t \rightarrow S_p^t = S_q^t$$

After combining Properties 1 and 4, we have Property 5:

Property 5. The metadata in the primary group P is consistent at time t if any two services in P receive the same message sequence at time t .

$$\forall p, q \in P \quad (\bar{Q}_p^t = \bar{Q}_q^t) \rightarrow G^t.$$

A group communication substrate provides a virtual synchronous environment for application processes. This means that every two processes that observe the same two consecutive membership changes receive the same set of regular multicast messages between the two changes. Regarding to the order of the regular messages received between the two changes, the group communication service provides total order communication. It guarantees that all messages are delivered in the same order to all group members. Combined with virtual synchrony and total order communication, a group communication service guarantees that any two services in the primary group P receive the same message sequence. According to Property 5, the metadata in the primary group P is consistent.

4. Total order broadcasting service

Section 3 shows that a total order broadcasting service is important to guarantee metadata consistency in a symmetric active/active service group. In this section, the services of a total order broadcasting are briefly discussed. We assume that there is a substrate layer providing basic broadcasting services. A typical parallel computing system comprises a set of processes that communicate via broadcasting messages. We assume that messages are uniquely identified through a pair (sender, counter).

4.1. Basic broadcasting services

In a distributed system of a set of machines, each machine has a sequence of events. An event is any operation executed on the machines, and thus sending or receiving a message is an event. A machine in the system uses a broadcasting service to send messages. A broadcast message is sent once by its source machine, and arrives at all target machines in the system at different times. The broadcasting service is responsible for the reliable delivery of messages. Internally, causal delivery order [28] of messages is guaranteed by the service. It is based on the relation ‘precedes’ (denoted by \rightarrow), which is defined as follows.

Definition 1. Assume that e_i and e_j are two events in such a distributed system. The transitive relation $e_i \rightarrow e_j$ (“happened before” relation) holds if any of the following conditions is satisfied:

- (1) e_i and e_j are two events on the same machine, and e_i comes before e_j .
- (2) e_i is the sending of a message m by one machine and e_j is the receipt of m by another machine.
- (3) There exists a third event e_k such that $e_i \rightarrow e_k$ and $e_k \rightarrow e_j$.

- Let m_i and m_j are two total order broadcasting messages:
 - (1) If m_i causally precedes m_j , all machines that deliver m_i and m_j deliver m_i before m_j .
 - (2) If m_i and m_j are concurrent, if machine p delivers m_i before m_j , then any machine q that belongs to the same partition with p delivers m_i before m_j .

Fig. 2. Total order broadcasting service definition.

The causal order for broadcast messages is defined as follows [23]:

(Causal order): If the broadcast of a message m_i causally precedes the broadcast of a message m_j , $m_i \rightarrow m_j$, then no machine delivers m_j before m_i .

Definition 2. Message m_i and message m_j are *concurrent* if m_i does not causally precede m_j and m_j does not causally precede m_i .

The basic broadcasting service receives the messages on the network. It keeps causal order of messages and delivers them to the fast delivery protocol. The broadcasting service does not guarantee the same delivery sequence of concurrent messages in the system.

Failure can be defined as deviations from correct behaviors. Some types of previously studied failure [23] include: crash failure, send omission failure, receive omission failure, and Byzantine failure. If a machine commits a failure, it is faulty; otherwise it is correct. The following properties [23] are guaranteed by the service:

(Validity): If a correct machine broadcasts a message m , then it eventually delivers m .

(Agreement): If a correct machine delivers a message m , then all correct processes eventually deliver m .

(Integrity): For any message m , every machine delivers m at most once, and only if m was previously broadcasted by *sender* (m).

4.2. Total order broadcasting

On top of the basic broadcasting service, totally ordered broadcasting extends the underlying causal order to a total order for concurrent messages.

(Total order): If two correct machines p and q both deliver message m_i and m_j , then p delivers m_i before m_j if and only if q delivers m_i before m_j .

The total order broadcasting provided by the system does not guarantee total order across multiple partitions. As long as partitions do not occur, all machines deliver the messages in the same total order. When a partition occurs, machines in the same partition continue to form the same total order. However, this may differ across partitions. The total order broadcasting service of the system is defined in Fig. 2.

5. Fast delivery protocol for total order broadcasting

Total order broadcasting is important for group communication services [13,7,15], but the agreement on the total order usually bears a cost of performance: a message is not delivered immediately after being received, until all the group members reach agreement on a single total order of delivery. Generally, the cost is measured as latency of totally ordered messages, from the point the message is ready to be sent, to the time it is delivered.

Traditionally three approaches are widely used to implement total ordering: sequencer, privilege-based, and communication history algorithms [15]. In sequencer algorithms, one machine is

responsible for ordering the messages on behalf of other machines in the group. Privilege-based algorithms rely on the idea that senders can broadcast messages only when they are granted the privilege to do so. For example, in a token-based algorithm [3], a token is rotated among machines in the same group, and one machine can only send messages while it holds the token. In communication history algorithms, total order messages can be sent by any machine at any time, without prior enforced order, and total order is ensured by delaying the delivery of messages, until enough information of communication history has been gathered from other machines.

These three types of algorithm have both advantages and disadvantages. Sequencer and privilege-based algorithms provide good performance when a system is relatively idle. However, when multiple machines are active and constantly send messages, the latency is limited by the time to circulate the token or produce the order number from the sequencer. Communication history algorithms have a post-transmission delay [15,18]. To collect enough information, the system has to wait for a message from each machine in the group, and then delivers the set of messages that do not causally follow any other, in a predefined order, for example, by sender ID. The length of the delay is set by the slowest machine to respond with a message. The post-transmission delay is most apparent when the system is relatively idle, and when waiting for response from all other machines in the group. In the worst case, the delay may be equal to the interval of heart beat messages from an idle machine. In contrast, if all machines produce messages and the communication in the group is heavy, the regular messages continuously form a total order, and the algorithm provides the potential for low latency of total order message delivery.

In a parallel computing system, multiple concurrent requests are expected to arrive simultaneously. A communication history algorithm is preferred to order requests among multiple machines, since such an algorithm performs well under heavy communication loads with concurrent requests. However, for relatively light load scenarios, the post-transmission delay is high.

In this section, we describe a *fast delivery* protocol to reduce this post-transmission delay. The fast delivery protocol provides the total order broadcasting service defined in Section 4.2. We assume that the protocol works on top of the basic broadcasting service described in Section 4.1. We first consider a static system of n machines, which means no failure of machines, no network partitions and re-merges, and no new machines. Those features will be considered in Section 5.3, in which we show how to extend the protocol to handle dynamic environments. The fast delivery protocol forms the total order by waiting for messages only from a subset of the machines in the group; thus it fast delivers total order messages. The protocol is implemented on top of the Transis [18] group communication system.

Each machine will not deliver any messages until it collects a message set from other machines in the group. The message set should contain enough information to guarantee totally ordered delivery. After a machine receives enough messages, it delivers the set of messages that do not causally follow any other, in a predefined order. Idle machines periodically broadcast heart beat messages with a predefined interval on behalf of other machines. Those heart beat messages will not be delivered, but are used by machines to determine the order of received messages.

5.1. Notation and definition

We define that a partition P consists of a group of machines $\{p_1, p_2, \dots, p_N\}$. We assume that each machine in the group P has a distinct ID. For a machine p , function $id(p)$ returns its ID. If the number of machines in the primary group P is N ,

$$\forall p, q \in P, \quad id(p), id(q) \in \{1, 2, \dots, N\}, \\ id(p) \neq id(q).$$

```

• When receiving a regular message  $m$  in machine  $p$ :
(1) if  $m$  is a new candidate message
    add  $m$  into candidate set  $M_p$ 
(2) if  $M_p \neq \phi$ 
    for all  $m_i \in M_p$  {
    if  $prefix(sender(m_i)) \subseteq senders(M_p)$ 
        add  $m_i$  into delivery set  $Md_p$ 
    }
(3) if  $Md_p \neq \phi$ 
    if causal order exists in  $Md_p$ 
        deliver messages in causal order
    for all concurrent messages  $m_j$  in  $Md_p$ 
        deliver in the order of  $id(sender(m_j))$ 
(4) if  $sender(m) \notin suffix(p)$ 
    return
    if message  $m$  is not a total order message
    return
    if there are messages waiting to be broadcast from the machine
     $p$ 
    return
    if  $\exists m_i \in M_p, id(sender(m_i)) = p$ 
    return
    otherwise, fast acknowledge  $m$ 

```

Fig. 3. Fast delivery protocol algorithm.

We associate with each machine $p \in P$ the functions $prefix$ and $suffix$ which are defined as follows:

- (1) $prefix(p) = \{q \mid \forall q \in P, id(q) < id(p)\}$
- (2) $suffix(p) = \{q \mid \forall q \in P, id(q) > id(p)\}$.

The input to the *fast delivery* protocol is a stream of causally ordered messages from the underlying broadcasting service. We define a function $sender(m)$ accepting an input parameter of any message m , and the function returns the ID of the machine sending this message m . For example, if a message m_i is sent by a machine p , then $sender(m_i) = p$. If a message m_i is delivered before a message m_j , $deliver(m_i) < deliver(m_j)$.

We define a *pending message* [17] to be a message that is received but the total order has not been agreed, and thus, it is not ready for delivery. A *pending message* that follows only delivered messages is called a *candidate message*. The set of concurrent candidate messages is called the *candidate set*. This is the set of messages that are considered for the next slot in the total order. For example, consider a system having five machines, $\{p_1, p_2, p_3, p_4, p_5\}$. After a certain time, there are no undelivered messages on any machines. Machine p_1 broadcasts a message m_{p_1} , and machine p_4 broadcasts a message m_{p_4} . All five machines receive both m_{p_1} and m_{p_4} , but none of them can deliver the two messages, because apart from the sending machines, no one knows if messages m_{p_2} and m_{p_3} are sent by p_2 and p_3 concurrently with m_{p_4} , or not. All machines should not deliver m_{p_1} and m_{p_4} until enough information is collected to determine the total order. The message set of m_{p_1} and m_{p_4} is called the candidate set, and messages m_{p_1} and m_{p_4} are called candidate messages. Let $M_p = \{m_1, m_2, \dots, m_k\}$ be the set of candidate messages in a machine p . We associate with M_p a function $senders$.

$$senders(M_p) = \{sender(m_i) \mid \forall m_i \in M_p\}.$$

Let Md_p be the set of messages ready to be delivered in a machine p such that $Md_p \subseteq M_p$. The set Md_p is called the *deliver set*.

5.2. The fast delivery protocol

The *fast delivery* protocol is symmetric, and we describe it for a specific machine p (see the pseudo code in Fig. 3). The basic concept

of the *fast delivery* protocol is to form the total order by waiting for messages only from a subset of the machines in the group. Assuming that a candidate message m is in *candidate set* M_p , we use the following delivery criterion to define what messages a machine has to wait for before delivering m :

- (1) Add m into *deliver set* Md_p when:

$$prefix(sender(m)) \subseteq senders(M_p).$$

- (2) Deliver the messages in set Md_p with the following order:

$$\forall m_i, m_j \in Md_p,$$

$$\text{if } m_i \rightarrow m_j$$

$$deliver(m_i) < deliver(m_j)$$

$$\text{if } m_j \rightarrow m_i$$

$$deliver(m_j) < deliver(m_i)$$

otherwise m_i and m_j are concurrent

$$id(sender(m_i)) < id(sender(m_j)) \longrightarrow deliver(m_i) < deliver(m_j).$$

With the same example as in Section 5.1, we explain how the protocol works. All five machines can deliver m_{p_1} immediately, because $prefix(p_1) = \phi$, and $senders(M_p) = \phi$. The five machines cannot deliver m_{p_4} , because $prefix(p_4) = \{p_1, p_2, p_3\}$ and $senders(M_p) = p_1$. Machines have to wait for messages from both p_2 and p_3 , but do not need to wait for a message from p_5 , because $p_5 \notin prefix(p_4)$.

Property 6. *With the fast delivery algorithm, if a machine p delivers m_i before m_j , then any machine q that belongs to the same partition with p delivers m_i before m_j , which means that total ordering is guaranteed in the same partition.*

Proof. (1) If there is a casual order between m_i and m_j , and if p delivers m_i before m_j , then m_i causally precedes m_j , so any machine q in the same partition delivers m_i before m_j .
(2) Otherwise, m_i and m_j are concurrent. If p delivers m_i before m_j , then $id(sender(m_i)) < id(sender(m_j))$. Any machine q in the same partition delivers m_i before m_j . \square

According to the protocol, if $prefix(sender(m)) \not\subseteq senders(M_p)$, the machine p has to wait for messages from other machines before delivering m . If any of those machines is idle, the waiting time could be up to the interval of heart beat messages. To speedup the delivery of m , idle machines should immediately acknowledge m on behalf of other machines. If a machine q receives a message m , and q is idle, q broadcasts a *fast acknowledgment* when

$$sender(m) \in suffix(q).$$

In the same example, if p_2 and p_3 are idle, they should fast acknowledge m_{p_4} , because $p_4 \in suffix(p_2)$, and $p_4 \in suffix(p_3)$. If p_5 is idle, it does not need to send a fast acknowledgment, because $p_4 \notin suffix(p_5)$.

Fast acknowledgment reduces the latency of message delivery; however, it injects more packets into the network. If communication is heavy, fast acknowledgment may burden the network and machines, and thus increase delivery latency. To reduce the cost, we define the following acknowledgment criterion:

(ACK) Fast acknowledge a message m from a machine q when:

- (1) message m is a total order message, and
- (2) there is no message waiting to be sent from the machine q , and
- (3) $\nexists m_j \in M_p, id(sender(m_j)) = q$.

Condition 1 is straightforward. Condition 2 means that if a machine is sending regular messages, it is not an idle machine, and the regular messages themselves are enough to form a total order. Condition 3 means that if a machine already sent a regular message, which is still in the set M_p , that message can be used to form a total order, without an additional acknowledgment. In the same example, if p_1 is idle after sending m_{p_1} , it does not need to send any acknowledgment (although $p_4 \in \text{suffix}(p_1)$), because m_{p_1} is still in M_p .

In a parallel system, when multiple concurrent requests arrive a machine simultaneously and the system is busy, conditions 2 and 3 could not be satisfied simultaneously, so no additional acknowledgments are injected into the network when communication is heavy.

5.3. Fast delivery protocol for dynamic systems

The fast delivery protocol operates on an asynchronous stream of causal order messages. So far, the protocol does not account for failures, network partitioning and re-merging, and joining machines. In this section, we show how to extend the protocol to handle the above issues in dynamic environments.

The fast delivery protocol is integrated into the group communication service to provide total order delivery of messages on top of the basic broadcasting service. We assume that the system contains a membership service, which maintains a view of the *current membership set* (CMS) consistent among all machines in the dynamic environment. When machines crash or disconnect, the network partitions and re-merges, or new machines join, the membership service of all connected machines must reconfigure and reach a new agreement on the CMS.

After a new agreement is reached, the membership service delivers a *view_change* event indicating a new configuration. All connected machines in the new configuration agree on the set of regular messages that belong to the previous membership, and must be delivered before the new *view_change* event. The fast delivery protocol is extended to define how to deliver such messages in the dynamic environment.

We assume that, after a new agreement on the CMS, the membership service notifies the fast delivery protocol with a special event. With such an event, the protocol gets the machine set, P_f , which belongs to the previous configuration, but is included in the new configuration. The new *prefix*(p) and *suffix*(p) are calculated based on the P_f :

- (1) $\text{prefix}(p) = \{q | \forall q \in P_f, \text{id}(q) < \text{id}(p)\}$
- (2) $\text{suffix}(p) = \{q | \forall q \in P_f, \text{id}(q) > \text{id}(p)\}$.

Using the algorithm described in Section 5, the set of regular messages that belong to the previous configuration are delivered before the *view_change* event with the new *prefix*(p) and *suffix*(p). Since a new CMS always completes within a finite delay of time, any total order message could be delivered within a limited time interval.

6. Symmetric active/active metadata service design

Conceptually, the symmetric active/active metadata service software architecture (Fig. 4) consists of several major parts, to handle client requests, update the global state, and manage membership of the service group. The current proof of concept prototype implementation uses the Transis [18] process group communication system in conjunction with the *fast delivery* protocol to provide total order and virtual synchrony, and the Parallel Virtual File System (PVFS) [11].

We provide basic metadata manipulation interfaces for clients. To balance workloads among multiple services, a client randomly

Table 1

Transaction control module locking table of the symmetric active/active metadata service.

	Read	Update	Write
Read			X
Update		X	X
Write	X	X	X

chooses a service to send a request. All client requests are sent to the request interface module. It interprets the requests, creates new jobs for them, then either dispatches the jobs directly to the basic metadata services module or requests Transis to broadcast them in total order. Jobs are first put into an active queue. The scheduler module chooses one active job to execute, until it is blocked by I/O operations and thus the job is put into the idle queue. After the I/O operations finish, the job is put back into the active queue and waits to be scheduled. The scheduling mechanism guarantees that a metadata service is not blocked by any I/O operation and multiple concurrent requests could be interleaved for improved throughput performance.

The basic metadata services module is responsible for updating local metadata and provides basic metadata management functions, such as create new object (file or directory), add a new entry into a directory, get attribute, and so on. Some client requests could be mapped to basic metadata services directly, such as get or set attribute, but some updating requests involve several basic metadata services and are also considered as atomic operations. For example, a create new file request involves three basic services: reading the parent directory to make sure no object has the same name, creating an object, and adding the handle of the new object into the parent directory. It is an atomic operation, because a failure of any step requires a roll back of all steps. The transaction control module is responsible for processing of such requests automatically, handling roll back if a failure occurs. It ensures that transactions are processed consistently across all service instances using process group communication services. All services make the same decision for a transaction, either submit, or rollback. The module coordinates transaction processing, and dispatches any real metadata operation to the basic services module.

Jobs are interleaved by the scheduler, but concurrent operations on the same objects are serialized by a locking service. The locking service provides three lock modes: read, write, and update. The incompatibility of the three modes is shown in Table 1. The update lock is designed to improve the performance of the transactions. A transaction first applies an update lock, without blocking other read requests, then upgrades to write mode only when operations modifying local objects are ready to be submitted. This design allows disabling the locking service if the parallel file system itself provides other means of locking at the client side or if POSIX file operation semantics are relaxed. Both may lead to further performance improvement.

The file handle space is managed by a dedicated module. Each metadata service allocates and releases file handles independently, but the file handle management must be consistent among all services of the group. The handle management module is responsible for allocating and releasing handles consistently for all services and maintaining the global state of the handle space.

The membership management module is responsible for maintaining integrity of the service group. Every time when new services join the group or current members leave the group, the module is notified with a view change message from Transis. The metadata is a global state that must be consistent across all services at any time, so the service group does not allow multiple partitions. Even if a network partition exists, the symmetric active/active metadata service group should only enable one primary group. Any service either belongs to a default primary group, or disables (fences) itself, until it rejoins the primary group. If a service

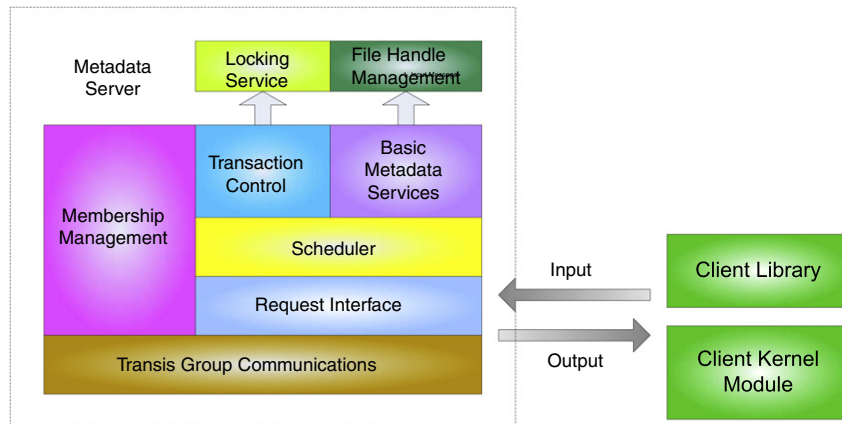


Fig. 4. Internal replication design of the symmetric active/active metadata service.

crashes, it is already disabled automatically. If a service leaves because of a network error, it must also stop responding to any client requests. If a client happens to connect a service not belonging to the primary group because of network partition, the client gets a negative response from the service; thus the client either tries to find other active services belonging to the primary group, or is notified with an operation failure event. In either case, the metadata is kept consistent. After joining the primary group, a service gets the current metadata service state from other members of the group, and thus updates its local copy. Since the view change messages are also totally ordered to request messages from clients, the current state of metadata obtained from other members is exactly consistent to the global state.

Read requests do not modify metadata except that, in POSIX compliant file systems, the last access time information is updated upon file read requests. The POSIX compliance of updating the last access time under read requests is relaxed in this design. The last access time is updated at the server handling the read request, but not instantly broadcasted to other servers. The update is put into a queue, and broadcasted to the group with later write requests. Any active service may handle read requests independently and locally (Fig. 5). However, write requests arriving at any metadata service have to be totally ordered by group communication services before being submitted to the basic metadata service (Fig. 6). A typical write request is processed by the following steps:

- (1) A client sends a request to the request interface.
- (2) Transis is requested to broadcast the state change message.
- (3) Transis delivers the state change message to a local scheduler.
- (4) The state change message is sent to the transaction control module.
- (5) The transaction control decomposes the state change messages.
- (6) The state change messages update the local metadata.
- (7) The operation results are returned to the request interface.
- (8) The service results are returned to clients.

7. Experimental results

To verify the above model, a proof of concept prototype for symmetric active/active metadata services has been implemented using the PVFS 2 [11] and deployed on the XTORC cluster at Oak Ridge National Laboratory, using up to four metadata services and 32 client nodes in various combinations for functional and performance testing. The compute nodes of the XTORC cluster are IBM IntelliStation M Pro series services. Individual nodes contain a Intel Pentium 2 GHz processor with 768 MB memory, and a 40 GB hard disk. All nodes are connected via Fast Ethernet (100 MBit/s

full duplex) switches. Although the Fast Ethernet is pretty slow, the network performance will not be the bottleneck of the system, since we only measure the read/write performance of metadata operations, which are very small messages (less than 1 KB in most requests). Fedora Core 5 has been installed as the operating system. Transis v1.03 with the *fast delivery protocol* is used to provide group communication services. Failures are simulated by unplugging network cables and by forcibly shutting down individual processes.

7.1. Benchmark

A set of micro- and macro-benchmark programs are used to evaluate the effectiveness of the fast delivery protocol and the active/active metadata service.

An MPI-based micro-benchmark was developed to study the latency and throughput of the group communication service and the metadata servers built on it. The benchmark has two testing modes. In mode A, multiple benchmark instances concurrently send messages to associated group communication servers, and then block until deliveries of all messages are confirmed. The latency of the group communication service is measured between the point the messages are sent out and the point the confirmations are delivered. In mode B, multiple benchmark instances concurrently send metadata requests to associated metadata servers, and wait for server responses. The blocked requests are sent for latency measurements, and a group of unblocked requests from each benchmark instance is sent for throughput measurements. Please bear in mind that the micro-benchmark only measures the latency/throughput of operations at metadata servers, not the operations of the entire file system.

Several macro-benchmarks, including PostMark [37], Iozzone [24], and BTIO from NAS PARALLEL BENCHMARKS (NPB) [33], were used to study the peculiarities of a metadata service. A tracing method is used to evaluate various standard file system workloads. Traces were collected by instrumenting a PVFS2 file system to log various operations in the metadata servers (MDSs) and the object-based storage devices (OSDs). The collected traces were analyzed offline to study the access patterns observed in the metadata services. The test bed consisted of one MDS, three OSDs, and multiple clients.

The PostMark benchmark was used to simulate heavy file system loads. PostMark is designed to create a large pool of continually changing files and to measure the transaction rates for a workload approximating a large Internet e-mail server. In this study, the initial number of files was set to 100. Files ranged between 1 MB to 64 MB in size. PostMark performed 1000 transactions on each file. Block sizes were 512 bytes and UNIX buffered I/O was used.

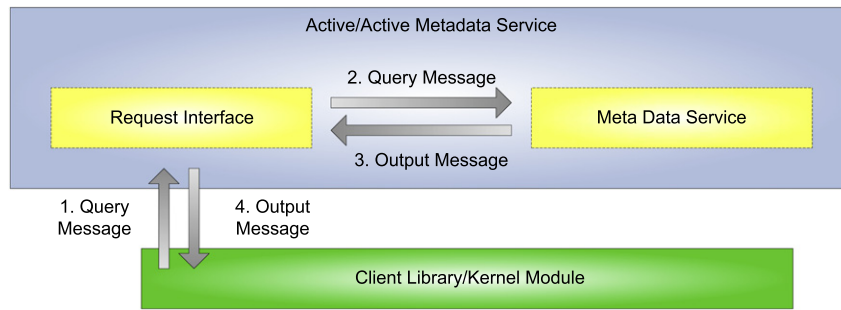


Fig. 5. Read request handling of the symmetric active/active metadata service.

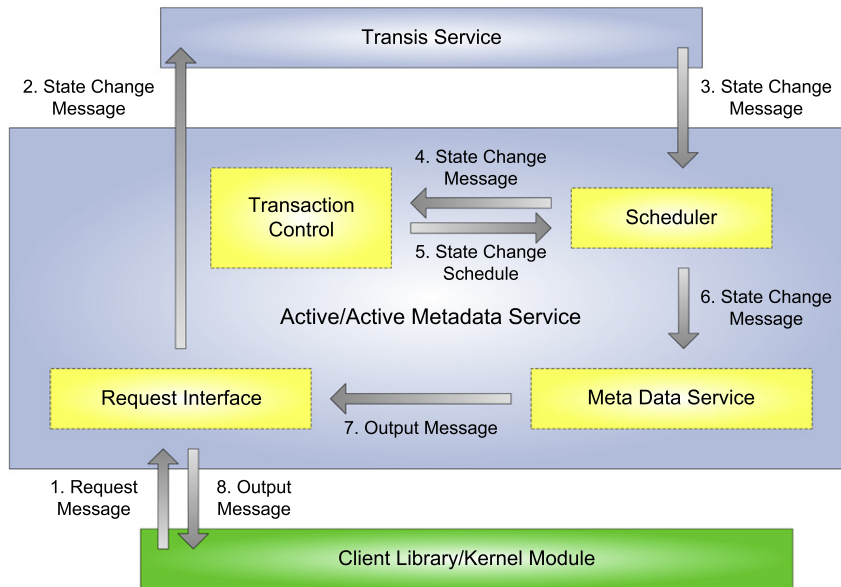


Fig. 6. Write request handling of the symmetric active/active metadata service.

iozone generates and measures a variety of file operations for determining a broad file system analysis. In this study, both sequential and random access patterns are simulated with read, write, re-read and re-write operations on a 4 GB file with various request sizes from 4 KB to 128 KB.

BTIO is an MPI-IO benchmark from NPB used for studying access patterns of parallel I/Os. It is compiled with OpenMPI [34] and ROMIO [41]. MPI-Full mode with collective I/O was enabled to examine the cost of the MDS and OSDs under concurrent I/O accesses from multiple clients.

7.2. Fast delivery protocol performance

At the first step, the effectiveness of the fast delivery protocol was examined. The mode A of the MPI-based micro-benchmark was used to send concurrent requests from multiple machines. The latency was measured with blocked requests, and an average latency was calculated from 100 requests of each machine. The results under various configurations were provided for comparison from one to eight machines. In the configuration of only one machine, the latency overhead mainly came from the processing cost of the group communication service. When the number of machines increased, additional overhead was introduced by the network communication and total order communication algorithm to reach agreement among machines. For each configuration, we measured the latency under both idle and busy systems. In an idle system, only a single machine sent requests. In a busy system, all machines sent concurrent requests.

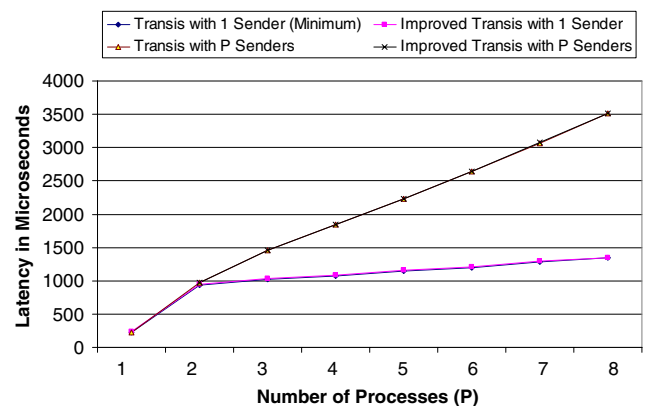


Fig. 7. Latency performance of the fast delivery protocol (maximum latency of Transis with one sender = heart beat interval, e.g., $\approx 500,000 \mu\text{s}$).

The fast delivery protocol was compared with the traditional communication history algorithm provided by the original Transis system. In an idle system, the post-transmission delay of the traditional communication history algorithm was apparent. The latency is a random variable, and in the worst case, it is equal to the interval of heart beat messages from an idle machine. A typical interval is in the gratitude of several hundred milliseconds, and the default value of Transis is 500 ms. To compare with the *fast delivery* protocol, Fig. 7 plots the minimum delay of Transis with one sender. In the figure, the latency of the plain communication history algorithm and the fast delivery protocol is almost identical,

because we compared to the minimum value of a random variable. The prototype shows that although the latency increased with the number of machines, the *fast delivery* protocol works well to keep the overall overhead far more acceptable and consistent. In an idle system, the latency of the new protocol is very close to the minimum delay of the Transis. In a busy system (Fig. 7 with P senders), the latency of the fast delivery protocol is almost the same as that of the traditional communication history algorithm, because the protocol held unnecessary acknowledgments. We found that when all machines sent concurrent requests, the fast delivery protocol did not acknowledge any broadcast, and the regular messages continuously formed the total order.

7.3. Metadata service failure handling

In the experiments, configuration changes including failures of metadata servers and new service joining were simulated. The initial configuration of the system consisted of one metadata server and multiple clients. Configuration changes were simulated at the time clients were randomly updating metadata. Excessive functional testing revealed correct behavior during normal system operation and in the case of single and multiple simultaneous failures. The functions monitored in the simulation included the following.

- (1) Correct behavior of metadata services in the process of configuration changes.
- (2) Consistence of metadata crossing all metadata servers in new configurations.
- (3) Any disruption of service or loss of service state in the process of configuration changes.

First, additional metadata servers were added into the service group. New servers were allowed to join the group, and served requests from clients right after they got the current metadata service state from other members of the group. A consistent metadata state was maintained in the group, for both old members and new members. There was no disruption of service or loss of service state in this process, and clients were not aware of changes of the configurations.

Then, one and more members were forced to leave the group, by unplugging network cables and by forcibly shutting down individual processes. Correct behavior of the metadata services was maintained and clients were not aware of any changes. The metadata state was consistent in the group before and after members left the group. No disruption of service or loss of service state occurred, while the metadata was maintained consistently at all services and high availability service was provided to clients.

7.4. Metadata service with a micro-benchmark

The proof of concept prototype shows a comparable latency and throughput performance. In the experiment, both the client and service caches were disabled to avoid interference. Mode B of the MPI-based micro-benchmark was used to send concurrent read/write requests from multiple clients. The latency and throughput of the original PVFS metadata service were compared with proposed symmetric active/active PVFS metadata service solution. The results under various configurations were provided for comparison between one, two, and four symmetric active/active metadata services. The latency was measured with blocked requests, and an average value was calculated from 100 requests of each client. The read latency was not provided, because read requests are independently handled by each active metadata service, and thus there was no difference to the original PVFS metadata service. The throughput was measured with unblocked requests. The total number of requests sent to the services was $5000 * N$, where

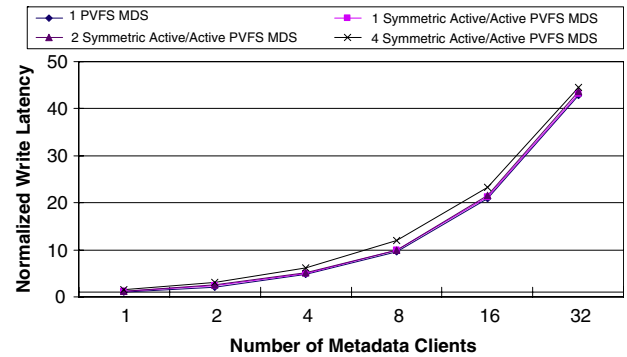


Fig. 8. Normalized write request latency performance of the symmetric active/active metadata service. The data is normalized to the latency of one PVFS server with one client.

N is the number of metadata services. Each client sent $\frac{5000 * N}{n}$ unblocked requests to services (n is the number of clients), and then waited for the completion of all requests. An average throughput was calculated in terms of requests/second.

Benefiting from the highly efficient total order service, the request latency of the prototype (Fig. 8) increases very conservatively from that of a single PVFS metadata server. The delay of a single PVFS metadata service was used as a baseline, and the data is normalized to the point of one PVFS service with one client. The latency of the baseline increases with the number of clients, and the normalized latency of active/active metadata service follows the same trend. The new design shows the same scalability as the baseline, since the latency difference remains small even with a large number of clients. It is consistent for both a small and a large number of clients. In the configuration of only one metadata service with symmetric active/active design, the latency overhead (compared to the baseline) mainly comes from processing cost of the Transis group communication service. When the number of metadata services increases, an additional overhead is introduced by the network communication and the total order communication algorithm to reach agreement among services. A service with two active/active servers is a practical configuration, because two servers could provide the necessary high availability with no disruption of service, but minimize the costs. In the configuration of two active/active servers, the latency overhead is very minor. The fast acknowledgment aggressively acknowledges the total order messages to reduce the latency of idle services when the number of clients is small. The protocol is smart enough to hold its acknowledgments when the network communication is heavy because more clients are involved.

The comparison of write throughput (Fig. 9) illustrates the trend that the overhead of virtual synchrony could be amortized with a large number of clients. At the point of one client, the configuration of two metadata servers introduces an overhead of 20%. If the number of clients increases to 32, the overhead drops to less than 5%. With a large number of clients, the regular messages from each server automatically form a total order without the overhead from additional acknowledgements.

In contrast, the read throughput (Fig. 10) increases linearly with the number of services. This is not surprising, because multiple services can process concurrent read requests simultaneously, a feature of the symmetric active/active replication architecture. With the same configuration of two servers, the improvement can be as high as 80%. In a metadata access pattern with realistic workload, the overheads suffering from update/write operations would be amortized with an improved scalability of read requests.

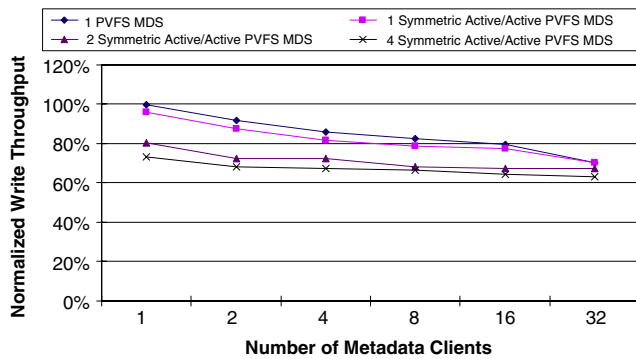


Fig. 9. Normalized write request throughput performance of the symmetric active/active metadata service. The data is normalized to the throughput of one PVFS server with one client.

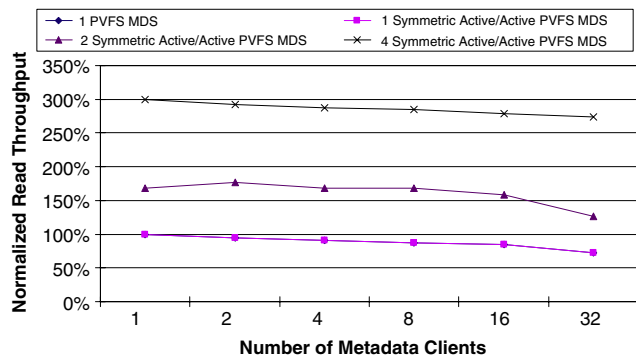


Fig. 10. Normalized read request throughput performance of the symmetric active/active metadata service. The data is normalized to the throughput of one server with one client.

7.5. Metadata service with macro-benchmarks

The use of symmetric active/active replication allows for no interruption of service upon failure of service nodes. Requiring virtual synchrony of all update/write operations adds an overhead to the “common path”. The prototype shows a very conservative overhead for write operations (Fig. 8), and shows that the read throughput increases linearly with the number of services (Fig. 10). The further study of access patterns observed in metadata services under various file system benchmarks (Section 7.1) could provide a view with realistic file system workloads of how the overall I/O performances would be impacted by applying the active/active metadata service and the possibility that the overheads suffered from update/write operations would be amortized with an improved scalability of read requests.

A file system operation initiated from clients consists of two types of sub-action: one is on the metadata server, and another is on OSDs. Correspondingly, the delay of such operations can be decomposed into the delay on the OSDs and the delay on the MDS. Different file system workloads have different access patterns, and the proportions of execution time on the MDS and on the OSDs are different. The collected traces from realistic file system workloads (Section 7.1) were analyzed by an in-house developed profiler. The overall execution time of these workloads was decomposed into the time spent on the OSDs and the time spent on the MDS (Fig. 11). The results indicate that, under most workloads, the execution time of the OSDs is a dominant figure of the overall I/O delay: the time spent on the metadata server is less than 1%. In those workloads, most file system operations occurred on OSDs. The comparison of execution time between the MDS and the OSDs illustrated that the overhead of metadata servers is a trivial component in the “common path” for most realistic workloads. As a

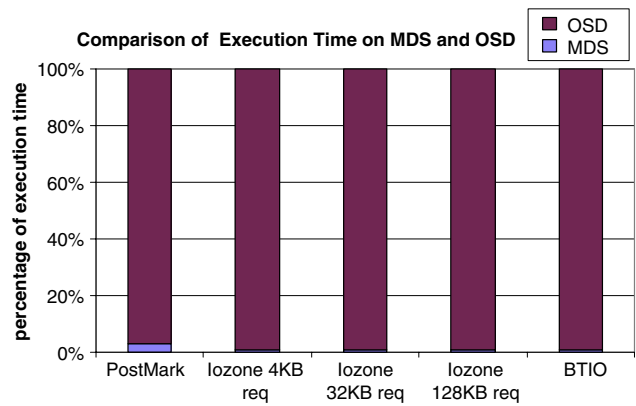


Fig. 11. Comparison of execution time of the MDS and the OSDs under various standard file system workloads. lozone 4 KB req, 32 KB req, and 128 KB req indicate the various request sizes in the study with lozone benchmark. MDS is the metadata server, and the OSD is the object-based storage device.

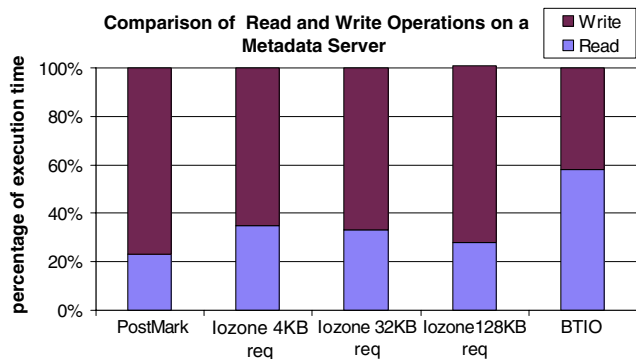


Fig. 12. Comparison of execution time spent on read operations and write operations on a metadata server under various standard file system workloads. lozone 4 KB req, 32 KB req, and 128 KB req indicate the various request sizes in the study with lozone benchmark. MDS is the metadata server, and OSD the object-based storage device.

consequence, overall I/O performances may not be impacted by applying the active/active metadata service.

Since multiple metadata services have the capability of processing concurrent read requests simultaneously, an analysis of the proportion of the execution time of write operations and read operations on a metadata server (Fig. 12) illustrates that the overheads suffered from update/write metadata operations may be amortized with an improved scalability of read requests. PostMark is the worst case: read operations accounts for 20% of total metadata execution time. Considering that the proof of concept prototype imposes a very conservative overhead for write operations, and the improvement of read operations could be as high as 80% with two metadata servers, a high availability of metadata services can be achieved with an acceptable performance trade-off even for workloads with a high amount of small I/O requests.

8. Related work

Past work on high availability for metadata services has primarily focused on the active/standby model. The PVFS metadata service, for example, can be deployed on two machines using active/standby and crosswise active/standby strategies involving a shared storage device and the heart beat mechanism [39]. Recent research in symmetric active/active replication for high performance system services [19,20] uses multiple redundant service nodes running in virtual synchrony [32]. In particular, the JOSHUA solution [49] for symmetric active/active HPC job and resource

management was a precursor to the research presented in this paper.

Previous research of distributed versioning [4,5] provided an efficient model to establish strong consistency (one-copy serializability) crossing multiple active replicas. Distributed versioning provides both the consistency guarantees of eager replication and the scaling properties of lazy replication. It does so by combining a novel concurrency control method based on explicit versions with conflict-aware query scheduling that reduces the number of lock conflicts, and content-aware scheduling that enable a lazy read-one, write-all replications. Both the distributed versioning model and the active/active model guarantee strong consistency among multiple active replicas, and they achieve high degrees of concurrency with the read-once, write-all scheme, but differences exist in how the two models implement the same total order at all replicas. Distributed versioning uses highly optimized schedulers and sequencers to assign distinct version numbers to replicas to guarantee the same total order, while the active/active model uses virtual synchrony and the fast delivery protocol to achieve the same total order.

Total order communication is important for symmetric active/active replication. Among the several algorithms to implement total ordering, there are three most used approaches: sequencer, privilege-based, and communication history [15]. In the sequencer algorithms [9,21,38,26], one machine is responsible for ordering the messages on behalf of other processes in the group. Privilege-based algorithms [14,3] rely on the idea that senders can broadcast messages only when they are granted the privilege to do so, and thus the total order is forced in the process of competition of the privilege. In communication history algorithms [18,17,16,1,8], total order is ensured by delaying the delivery of messages, until enough information about the communication history has been gathered from other machines. The agreement on the total order usually bears a cost of performance: the latency from the point the message is ready to be sent to the time it is delivered. Early delivery algorithms [17,8] reduce the latency by reaching agreement with a subset of the machines in the group. Optimal delivery algorithms [25,51] deliver messages before the total order is determined, but notify the applications and cancel the delivery if the final order is different from the delivered order.

Metadata is very important for file systems. Researchers have developed many algorithms to efficiently and reliably manage metadata. An approach called Lazy Hybrid metadata management for high performance object-based storage systems [10] combines the best aspects of two managing metadata techniques: directory subtree partitioning and pure hashing. Zhu et al. [54] propose a hierarchical metadata management scheme. A dynamic subtree partitioning and adaptive metadata management system [52] is proposed to service a petabyte-scale distributed file system, using a dynamic subtree technique to distribute the workload while maximizing overall scalability. To solve the metadata consistency problem, a metadata snapshotting [46] is presented to provide system availability at very little cost. Journal-based metadata and multi-version b-trees [47] are used in a comprehensive versioning file system (CVFS) to reduce the space utilization without decreasing the performance of data accessing. Two methods for improving metadata operations, journaling and soft updates, are explored [44] to improve the performance of metadata operations.

Previous file systems have distributed and replicated metadata and user data to improve availability. The XFS file system [6] distributes metadata into multiple managers across the system on a per-file granularity by utilizing a new serviceless management scheme. Furthermore, location independence provides high availability by allowing any machine to take over the responsibilities of a failed component after recovering its state from the redundant log-structured storage system. The active/standby model is

used in XFS to organize a redundant storage system. The Frangipani file system [48] uses the large, sparse disk address space of the substrate Petal storage system [30] to simplify its data structures. The data and metadata of Frangipani are stored and managed on top of the virtual address space provided by Petal, similar to traditional file systems on top of the address space of hard disks, but the real data is physically distributed to multiple Petal storage services. High availability of both user data and metadata is provided by a replication based redundancy scheme called chained de-clustering of the Petal system.

Various research efforts in file systems have shown that total ordering can be used to provide high availability. The Deceit file system [45] behaves like a plain NFS service. The deceit services are interchangeable and collectively provide the illusion of a single service to any clients. It uses the ISIS [9] distributed programming environment for all totally ordered communication and process group management. Non-volatile replicas of each file are stored on a subset of the file services.

Recent research in providing service redundancy has focused on practical solutions for solving the *Byzantine generals problem* [29], where malicious attacks and software errors result in incorrect process group behavior. These approaches go beyond the *fail-stop model*, which assumes that system components, such as services, nodes, or links, fail by simply stopping. Byzantine Fault Tolerance with Abstract Specification Encapsulation (BASE) [40] is a communication library for state machine replication. It uses abstraction to reduce the cost of Byzantine fault tolerance and to improve its ability to mask software errors. Using BASE, each replica can be repaired periodically using an abstract view of the state stored by correct replicas. Furthermore, each replica can run distinct or nondeterministic service implementations, which reduces the probability of common mode failures. Prototype implementations for a networked file system [12] and an object-oriented database [50] suggest that the technique can be used in practice with a modest amount of adaptation and with comparable performance results.

9. Conclusions

This paper has presented recent research in symmetric active/active metadata services as a generic approach for highly available cluster storage systems. This concept provides a virtually synchronous environment for high availability without any interruption of service and without any loss of state. It guarantees the safety of global state updating by utilizing group communication services and total order broadcasting. The number of services in the service group is variable at runtime. The internal scheduler improves the system performance by interleaving concurrent requests, and the built-in transaction control mechanism provides atomic services and guarantees data consistency in the case of operation failures.

A fast delivery protocol is illustrated to reduce the latency of ordering messages. The protocol optimizes the total ordering process by waiting for messages only from a subset of the machines in the group. The protocol performs well for both idle and active services. Furthermore, the fast acknowledgment aggressively acknowledges total order messages to reduce the latency when some services are idle. The protocol is smart enough to hold its acknowledgments when the network communication is heavy.

Both the functional and the performance test results are presented with comprehensive experiments under various system configurations. The results show that, for write requests, the overhead of latency and throughput increases with the number of services, but is still acceptable for typical distributed storage systems. The throughput of read requests increases linearly with the number of services. The experimental results show that high availability of metadata services can be achieved without interruption of service and with an acceptable performance trade-off using the active/active metadata service solution.

Acknowledgments

This work was sponsored in part by the Office of Advanced Scientific Computing Research, U.S. Department of Energy. The work at Tennessee Tech University was sponsored by the Laboratory Directed Research and Development Program of ORNL, by the U.S. National Science Foundation under Grant Nos. CNS-0617528 and CNS-0720617, and by the Office of Research of Tennessee Technological University. It was performed in part at Oak Ridge National Laboratory (ORNL), which is managed by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725. The authors would like to thank the anonymous reviewers and the Elsevier editor for their valuable feedback to improve the quality of this article.

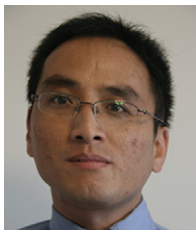
This paper extends the concepts and results presented in two earlier papers at the 19th International Conference on Parallel and Distributed Computing and Systems (PDCS) 2007 [35] and the 16th International Conference on Computer Communications and Networks (ICCCN) 2007 [36].

References

- [1] Marcos Kawazoe Aguilera, Robert E. Strom, Efficient atomic broadcast using deterministic merge, in: Proceedings of the 19th Annual ACM Symposium on Principles of Distributed Computing, PODC, Portland, OR, USA, July 16–19, 2000, ACM Press, New York, NY, USA, 2000, pp. 209–218.
- [2] Altair Engineering, Troy, MI, USA. PBS Pro for Cray Computing Platforms. Available at http://www.pbsgridworks.com/PBS/pdfs/PBSPro_Cray.pdf, 2007.
- [3] Yair Amir, Louise E. Moser, Peter M. Melliar-Smith, Deborah A. Agarwal, Paul W. Ciarfella, The Totem single-ring ordering and membership protocol, *ACM Transactions on Computer Systems (TOCS)* 13 (4) (1995) 311–342.
- [4] C. Amza, A. Cox, W. Zwaenepoel, Distributed versioning: Consistent replication for scaling back-end databases of dynamic content web sites, in: Proceedings of the ACM/IFIP/Usenix Middleware Conference, June 2003.
- [5] C. Amza, A. Cox, W. Zwaenepoel, A comparative evaluation of transparent scaling techniques for dynamic content servers, in: Proceedings of the IEEE International Conference on Data Engineering, ICDE, April 2005.
- [6] Thomas E. Anderson, Michael D. Dahlin, Jeanna M. Neefe, David A. Patterson, Drew S. Roselli, Randolph Y. Wang, Serverless network file systems, *ACM SIGOPS Operating Systems Review (OSR)* 29 (5) (1995) 109–126.
- [7] Roberto Baldoni, Stefano Cimmino, Carlo Marchetti, Total order communications: A practical analysis, in: Proceedings of the 5th European Dependable Computing Conference, EDCC, Budapest, Hungary, April 20–22, 2005, in: Lecture Notes in Computer Science, vol. 3463, Springer Verlag, Berlin, Germany, 2005, pp. 38–54.
- [8] Ziv Bar-Joseph, Idit Keidar, Nancy Lynch, Early-delivery dynamic atomic broadcast, in: Proceedings of the 16th International Conference on Distributed Computing, DISC, Toulouse, France, October 28–30, 2002, in: Lecture Notes in Computer Science, vol. 2508, Springer Verlag, Berlin, Germany, 2002, pp. 1–16.
- [9] Kenneth P. Birman, Robbert van Renesse, Reliable Distributed Computing with the Isis Toolkit, IEEE Computer Society, 1994.
- [10] Scott A. Brandt, Ethan L. Miller, Darrell D.E. Long, Lan Xue, Efficient metadata management in large distributed storage systems, in: Proceedings of the 20th IEEE/19th NASA Goddard Conference on Mass Storage Systems and Technologies, MSST, San Diego, CA, USA, April 7–10, 2003, IEEE Computer Society, 2003, pp. 290–298.
- [11] Philip H. Carns, Walter B. Ligon III, Robert B. Ross, Rajeev Thakur, PVFS: A parallel file system for Linux clusters, in: Proceedings of the 4th Annual Linux Showcase and Conference 2000, Atlanta, Georgia, October 10–14, 2000 pp. 317–327.
- [12] Miguel Castro, Barbara Liskov, Practical Byzantine fault tolerance and proactive recovery, *ACM Transactions on Computer Systems (TOCS)* 20 (4) (2002) 398–461.
- [13] Gregory V. Chockler, Idit Keidar, Roman Vitenberg, Group communication specifications: A comprehensive study, *ACM Computing Surveys (CSUR)* 33 (4) (2001) 427–469.
- [14] Flaviu Cristian, Shivakant Mishra, Guillermo Alvarez, High-performance asynchronous atomic broadcast, *Distributed Systems Engineering* 4 (2) (1997) 109–128.
- [15] Xavier Défago, André Schiper, Péter Urbán, Total order broadcast and multicast algorithms: Taxonomy and survey, *ACM Computing Surveys (CSUR)* 36 (4) (2004) 372–421.
- [16] Danny Dolev, Idit Keidar, Totally ordered broadcast in the face of network partitions, in: Dependable Network Computing, 2000, pp. 51–75 (Chapter 3).
- [17] Danny Dolev, Shlomo Kramer, Dalia Malki, Early delivery totally ordered multicast in asynchronous environments, in: Proceedings of the 23rd Annual International Symposium on Fault-Tolerant Computing, FTCS, Toulouse, France, June 22–24, 1993, IEEE Computer Society, 1993, pp. 544–553.
- [18] Danny Dolev, Dalia Malki, The Transis approach to high availability cluster communication, *Communications of the ACM* 39 (4) (1996) 64–70.
- [19] Christian Engelmann, Stephen L. Scott, Concepts for high availability in scientific high-end computing, in: Proceedings of the High Availability and Performance Workshop, HAPCW 2005, in conjunction with the Los Alamos Computer Science Institute, LACSI Symposium 2005, Santa Fe, NM, USA, October 11, 2005.
- [20] Christian Engelmann, Stephen L. Scott, Chokchai Leangsuksun, Xubin He, Symmetric active/active high availability for high-performance computing system services, *Journal of Computers (JCP)* 1 (8) (2006) 43–54.
- [21] M. Frans Kaashoek, Andrew S. Tanenbaum, An evaluation of the amoeba group communication system, in: Proceedings of the 16th IEEE International Conference on Distributed Computing Systems, ICDCS, Hong Kong, June 27–30, 1996, IEEE Computer Society, 1996, pp. 436–447.
- [22] Ibrahim Haddad, Chokchai Leangsuksun, Stephen L. Scott, Tong Liu, HA-OSCAR: Towards highly available Linux clusters, *Linux World Magazine* (March) (2004).
- [23] Vassos Hadzilacos, Sam Toueg, A modular approach to fault-tolerant broadcasts and related problems. Technical Report 94-1425, Department of Computer Science, Cornell University, Ithaca, NY, USA, 1994.
- [24] Iozone file system benchmark, in: <http://www.iozone.org/>.
- [25] Bettina Kemme, Fernando Pedone, Gustavo Alonso, Schiper Schiper, Matthias Wiesmann, Using optimistic atomic broadcast in transaction processing systems, *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 15 (4) (2003) 1018–1032.
- [26] Jongsung Kim, Cheeha Kim, A total ordering protocol using a dynamic token-passing scheme, *Distributed Systems Engineering* 4 (2) (1997) 87–95.
- [27] Leslie Lamport, Using time instead of timeout for fault-tolerant distributed systems, *ACM Transactions on Programming Languages and Systems* 6 (2) (1984) 254–280.
- [28] Leslie Lamport, Time, clocks, and the ordering of events in a distributed system, *Communications of the ACM* 21 (7) (1978) 558–565.
- [29] Leslie Lamport, Robert E. Shostak, Marshall C. Pease, The Byzantine generals problem, *ACM Transactions on Programming Languages and Systems (TOPLAS)* 4 (3) (1982) 382–401.
- [30] Edward K. Lee, Chandramohan A. Thekkath, Petal: Distributed virtual disks, in: Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS, Cambridge, MA, USA, October 1–5, 1996, ACM Press, New York, NY, USA, 1996, pp. 84–92.
- [31] Lustre file system - high-performance storage architecture and scalable cluster file system, White Paper, Sun Microsystems, Inc, Santa Clara, CA, USA, 2007, Available at http://www.sun.com/software/products/lustre/docs/lustrefilesystem_wp.pdf.
- [32] Louise E. Moser, Yair Amir, Peter M. Melliar-Smith, Deborah A. Agarwal, Extended virtual synchrony, in: Proceedings of the 14th IEEE International Conference on Distributed Computing Systems, ICDCS, Poznan, Poland, June 21–24, 1994, IEEE Computer Society, 1994, pp. 56–65.
- [33] Nas parallel benchmarks, in: <http://www.nas.nasa.gov/Resources/Software/npb.html>.
- [34] Open mpi: Open source high performance computing, in: <http://www.open-mpi.org/>.
- [35] Li Ou, Christian Engelmann, Xubin He, Xin Chen, Stephen L. Scott, Symmetric active/active metadata service for highly available cluster storage systems, in: Proceedings of the 19th IASTED International Conference on Parallel and Distributed Computing and Systems, PDCS, Cambridge, MA, USA, November 19–21, 2007, ACTA Press, Calgary, AB, Canada, 2007.
- [36] Li Ou, Xubin He, Christian Engelmann, Stephen L. Scott, A fast delivery protocol for total order broadcasting, in: Proceedings of the 16th IEEE International Conference on Computer Communications and Networks, ICCCN, Honolulu, HI, USA, August 13–16, 2007, IEEE Computer Society, 2007.
- [37] Postmark filesystem performance benchmark, in: <http://www.netapp.com>.
- [38] Michael K. Reiter, Distributing trust with the Rampart toolkit, *Communications of the ACM* 39 (4) (1996) 71–74.
- [39] Alan L. Robertson, The evolution of the Linux-HA project, in: Proceedings of the UKUUG LISA/Winter Conference – High-Availability and Reliability – 2004, Bournemouth, UK, June 21, 2004.
- [40] Rodrigo Rodrigues, Miguel Castro, Barbara Liskov, BASE: Using abstraction to improve fault tolerance, *ACM SIGOPS Operating Systems Review (OSR)* 35 (5) (2001) 15–28.
- [41] Romio: A high-performance, portable mpi-io implementation, in: <http://www-unix.mcs.anl.gov/romio/>.
- [42] Drew Roselli, Jacob R. Lorch, Thomas E. Anderson, A comparison of file system workloads, in: Proceedings of the USENIX Annual Technical Conference, USENIX, San Diego, CA, USA, June 18–23 2000, USENIX Association, Berkeley, CA, USA, 2000, pp. 41–54.
- [43] Fred B. Schneider, Implementing fault-tolerant services using the state machine approach: A tutorial, *ACM Computing Surveys* 22 (4) (1990) 299–319.
- [44] Margo I. Seltzer, Gregory R. Ganger, M. Kirk McKusick, Keith A. Smith, Craig A.N. Soules, Christopher A. Stein, Journaling versus soft updates: Asynchronous meta-data protection in file systems, in: Proceedings of the USENIX Annual Technical Conference, USENIX, USENIX Association, Berkeley, CA, USA, 2000, pp. 18–23.
- [45] Alexander Siegel, Kenneth P. Birman, Keith Marzullo, Deceit: A flexible distributed file system. Technical Report 89-1042, Department of Computer Science, Cornell University, Ithaca, NY, USA, 1989.
- [46] Livio B. Soares, Orran Y. Krieger, Dilma Da Silva, Meta-data snapshotting: A simple mechanism for file system consistency, in: Proceedings of the International Workshop on Storage Network Architecture and Parallel I/O, SNAPI, 2003, in conjunction with the 12th International Conference on Parallel

Architectures and Compilation Techniques, PACT, 2003, New Orleans, LA, USA, September 28, 2003, ACM Press, New York, NY, USA, 2003, pp. 41–52.

- [47] Craig A.N. Soules, Garth R. Goodson, John D. Strunk, Gregory R. Ganger, Metadata efficiency in versioning file systems, in: Proceedings of the 2nd USENIX Conference on File and Storage Technologies, FAST, San Francisco, CA, USA, March 31 – April 2, 2003, USENIX Association, Berkeley, CA, USA, 2003, pp. 43–58.
- [48] Chandramohan A. Thekkath, Timothy Mann, Edward K. Lee, Frangipani: A scalable distributed file system, *ACM SIGOPS Operating Systems Review (OSR)* 31 (5) (1997) 224–237.
- [49] Kai Uhlemann, Christian Engelmann, Stephen L. Scott, JOSHUA: Symmetric active/active replication for highly available HPC job and resource management, in: Proceedings of the 8th IEEE International Conference on Cluster Computing (Cluster), Barcelona, Spain, September 25–28, 2006, IEEE Computer Society, 2006.
- [50] Ben Vandiver, Hari Balakrishnan, Barbara Liskov, Sam Madden, Tolerating Byzantine faults in transaction processing systems using commit barrier scheduling, in: Proceedings of the 21st ACM SIGOPS Symposium on Operating System Principles, SOSP, Stevenson, WA, USA, October 14–17, 2007, ACM Press, New York, NY, USA, 2007, pp. 59–72.
- [51] Pedro Vicente, Luis Rodrigues, An indulgent uniform total order algorithm with optimistic delivery, in: Proceedings of the 21th IEEE Symposium on Reliable Distributed Systems, SRDS, Suita, Japan, October 13–16, 2002, IEEE Computer Society, 2002, pp. 92–101.
- [52] Sage A. Weil, Kristal T. Pollack, Scott A. Brandt, Ethan L. Miller, Dynamic metadata management for petabyte-scale file systems, in: Proceedings of the ACM/IEEE Conference on High Performance Computing, Networking and Storage, SC, Pittsburgh, PA, USA, November 6–12, 2004, IEEE Computer Society, 2004.
- [53] Andy B. Yoo, Morris A. Jette, Mark Grondona, SLURM: Simple Linux utility for resource management, in: Proceedings of the 9th International Workshop on Job Scheduling Strategies for Parallel Processing, JSSPP, Seattle, WA, USA, June 24, 2003, in: Lecture Notes in Computer Science, vol. 2862, Springer Verlag, Berlin, Germany, 2003, pp. 44–60.
- [54] Yifeng Zhu, Hong Jiang, J. Wang, Hierarchical Bloom Filter (HBA): A novel, scalable metadata management system for large cluster-based storage, in: Proceedings of the 6th IEEE International Conference on Cluster Computing, Cluster, San Diego, CA, USA, September 20–23, 2004, IEEE Computer Society, 2004, pp. 165–174.



Xubin He received his PhD degree in electrical engineering from the University of Rhode Island, USA, in 2002 and his BS and MS degrees in computer science from Huazhong University of Science and Technology, China, in 1995 and 1997, respectively. He is currently an associate professor in the Department of Electrical and Computer Engineering at Tennessee Technological University and supervises the Storage Technology Architecture Research (STAR) Laboratory. His research interests include computer architecture, storage systems, virtualization, and high availability computing. He received the Ralph E. Powe Junior Faculty Enhancement Award in 2004 and the TTU Chapter Sigma Xi Research Award in 2005. He is a senior member of the IEEE and the IEEE Computer Society.



Li Ou received his PhD Degree in Computer Engineering from Tennessee Technological University in December 2006, and his BS degree in information technology, and MS Degree in Computer Science from the University of Electronics Science & Technology of China, in 1997 and 2003, respectively. He is currently a researcher at DELL. His research interests include computer architecture, storage and I/O systems, and high performance cluster computing.



Dr. Christian Engelmann is a Research and Development Associate at Oak Ridge National Laboratory (ORNL). He holds a PhD and an MSc in Computer Science from the University of Reading, UK, and an MSc in Computer Systems Engineering from the University of Applied Sciences, Berlin, Germany. His research within the last 9 years at ORNL aims at computer science challenges for next-generation extreme-scale high-performance computing (HPC) system software, such as dependability, scalability, and portability.



Xin Chen received his BS degree in Mechanical Engineering from Shanghai University, Shanghai, China, and he received his MS degree in Electrical Engineering from Tennessee Technological University in 2007. He is currently working toward the PhD degree in the Department of Electrical and Computer Engineering, Tennessee Technological University. His research interests include distributed file systems, networking, and performance and availability evaluation of distributed systems.



Dr. Stephen L. Scott is a Senior Research Scientist and team leader of the System Research Team in the Computer Science Group of the Computer Science and Mathematics Division at the Oak Ridge National Laboratory, Oak Ridge, TN, USA. Dr. Scott's research interest is in experimental systems with a focus on high performance distributed, heterogeneous, and parallel computing. He is a founding member of the Open Cluster Group and Open Source Cluster Application Resources (OSCAR). Dr. Scott has published over 100 peer-reviewed papers in the areas of parallel, cluster and distributed computing and holds both a PhD and an MS in computer science.