

# A Distributed Paging RAM Grid System for Wide-Area Memory Sharing<sup>1</sup>

Rui Chu<sup>†</sup>, Nong Xiao<sup>†</sup>, Yongzhen Zhuang<sup>‡</sup>, Yunhao Liu<sup>‡</sup>, and Xicheng Lu<sup>†</sup>

<sup>†</sup>National Key Laboratory for Parallel and Distributed Processing,  
ChangSha, HuNan, China  
{rchu, nongxiao, xclu}@nudt.edu.cn

<sup>‡</sup>Hong Kong University of Science and Technology  
Clear Water Bay, Kowloon, Hong Kong  
{csyz, liu}@cs.ust.hk

## Abstract

*Memory-intensive applications often suffer from the poor performance of disk swapping when memory is inadequate. Remote memory sharing schemes, which provide a remote memory that is faster than the local hard disk, are able to improve the performance of such applications. Due to the limitation of being applicable within single clusters only, however, most of the previous remote memory mechanisms, such as the network memory scheme, fail to be extendable into a large scale, distributed, heterogeneous, and dynamic environment. In this work, we propose a service-oriented grid memory sharing scheme, Distributed Paging RAM Grid (DPRG). We study the properties and criteria of large scale memory sharing, and then design major operations and optimizations to fit the usage of grid systems. We collect trace from our grid environment, and evaluate DPRG through comprehensive trace-driven simulations. Results show that DPRG significantly outperforms existing remote memory sharing schemes and supports grid computing applications effectively.*

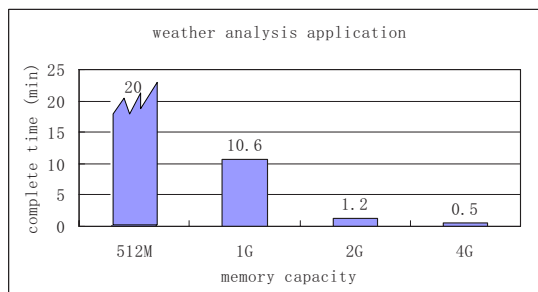
## 1. Introduction

Many scientific computing applications demand significant amounts of memory, and are usually run on supercomputers with large memory capacities. A typical example is a meteorological analysis application from the China Meteorological Administration. As shown in Figure 1, running a meteorological analysis task with a peak memory demand of 1.9G takes more than 20 minutes – the program abnormally exits due to low swap

space – on a workstation with 512M of physical memory, but completes in about 30 seconds when there is 4G of physical memory.

The tradeoff between memory capacity and completion time exists in many scientific computing applications. For a complex task with a high memory demand, scientists are forced to lower the computing precision to decrease the memory demand in order to obtain results on time. Both page-faults and insufficient disk caches can cause the performance to drop. The ultimate problem is the gap in speed and capacity between the memory and hard disk [1].

Clusters that use network memory mechanisms to share memory between cluster nodes have long been considered to be the best computing facilities for memory-intensive applications [2]. However, the cluster size is often restricted, and sometimes all of the cluster nodes are lacking in memory. Moreover, memory sharing in clusters may cause interior network congestion which



**Figure 1. Tradeoff between memory capacity and complete time.**

<sup>1</sup> The work was partially supported by the National Basic Research Program of China (973) under Grant No.2005CB321801, No. 2003CB317008 and the National Natural Science Foundation of China under Grant No. 90412011, No. 60573135 and No. 60573140, Hong Kong RGC DAG 05/06. EG44.

has the following inevitable problems. First, the large amount of traffic caused by transferring cluster memory greatly increases the access time of cluster memory. Second, when one of the cluster nodes running a parallel job encounters a memory shortage, the overall performance drops significantly due to the synchronization overhead caused by this slower node. In the above scenarios, the overhead of using cluster memory is even greater than that of using a local hard disk [3].

Grid and P2P computing has been an attractive distributed computing paradigm over wide-area networks[4-7]. An abundant amount of idle memory connected with high speed networks is scattered throughout the Internet. Some are found to have moderate speeds and capacities between the local memory and disk. This motivates us to the idea of utilizing remote memory resources beyond the cluster. The existing network memory schemes[2], which were originally designed for clusters, cannot work well under wide-area distributed, heterogeneous, dynamic, and autonomous conditions, and thus, cannot be directly applied to grid memory sharing.

In this paper, we propose a new mechanism, Distributed Paging RAM Grid (DPRG). We call the remote memory resources on the Internet grid memory. In our approach, the congestion of memory traffic is alleviated by distributing the traffic-load over different network paths. Thus, single-node over-loading occurs less often since abundant memory resources are always available on the Internet. DPRG adopts a service-oriented architecture[8] and provides memory service based on the wide-area distributed computing technology. To the best of our knowledge, we are the first to propose and design a RAM Grid system.

The rest of this paper is organized as follows. In Section 2 we introduce the design criteria and system background. In Section 3 we discuss the DPRG architecture. Section 4 provides the simulation methodology, and Section 5 presents the experimental results and analysis. Section 6 describes the related work. Section 7 concludes the paper and points to future work.

## 2. Overview

According to Patterson's observation [9], the latency of hard disks with different rotation speeds and seek times ranges from 5 to 35 *ms*, while the network latency of LAN or WAN can be as low as several hundred  $\mu$ s. It is indicated that cluster memory is faster than disk memory by a factor of several hundred. Though its performance varies due to different queuing and transmission delays along the traveling path, grid

memory in the same city or campus can be several dozen times faster than disk memory.

The advantages of RAM Grid are more than the high access speed of a single remote memory compared with disk speed. They include the capability to use Internet memory services and improve the overall performance of a computational task, the release of interior traffic congestion, and high performance even when the application memory demand is tremendously high.

### 2.1. System Assumption

We first make several reasonable assumptions in the design of a prototype of a RAM Grid system - DPRG, through which we separate sub-problems (mutual trust) and throw away the trivia (zero disk cache).

**Security and mutual trust:** In DPRG, all nodes joined in the system are considered to be trustworthy. As the first step in the design of the RAM Grid system, we only focus on the basic system design and the performance evaluation. Security and trust mechanisms are necessary to ensure safe transfer and execution. This problem is separated from system core functions. Presently, we assume that data transmission is secure and all nodes in the system are trustworthy.

**Zero disk cache:** Each hard disk includes an inner disk cache, whose size is usually in the range of several megabytes. The speed of the disk cache is close to the memory speed, and much faster than the hard disk speed, but its effect in performance upgrade is not as significant as that of free memory of the same size. Thereby, for simplicity we can treat the memory and inner disk cache as equivalent devices with the same access speed. If a node has a memory of size  $X$  and an inner disk cache of size  $Y$  (usually  $X \gg Y$ ), we simply regard it as a node having a memory of size  $X+Y$ .

### 2.2. Criteria

We extract three system criteria, which are the basic premises of DPRG.

**Single service offering:** Each grid node can only offer service to one user. For example, if node  $x$  is storing page frames for node  $y$ , it cannot store page frames for any other node. Though the memory utilization is higher if a grid node can work for more users simultaneously, this one-to-many approach has the following problems. First, since the remote memory is shared by several users, each memory service user is not certain how much free memory the remote node can offer in the future. It can only obtain this information at each turn of the memory request, which decreases the system performance. Second, when multiple users are accessing their page frames from the same node simultaneously, they suffer

from queuing delays. Therefore, single service offering is reasonable since the memory resources are plentiful in a RAM Grid. It is tolerable to have a little memory waste in each node for the sake of fewer adjustments and less queuing overheads.

**Exponential request:** When a user needs memory, it requests  $M, 2M, 4M, \dots, 2^{n-1}M$  of remote memory gradually until its need is satisfied, where  $M$  is the amount of local memory of the node. There is a tradeoff between the time consumption and the memory waste in this gradually increasing request. A small memory in each round means more time spent for repeating resource discovery, while a large memory in each round means memory wasted. Assuming  $P$  is the total amount of the requested memory, and  $t$  is the time spent for each round of resource discovery, exponential request is a middle course that can achieve moderate time consumption,

$$\left\lceil \log_2 \frac{P+M}{M} \right\rceil * t, \quad (1)$$

and moderate memory waste,

$$M * (2^{\left\lceil \log_2 \frac{P+M}{M} \right\rceil} - 1) - P. \quad (2)$$

### 3. System Design

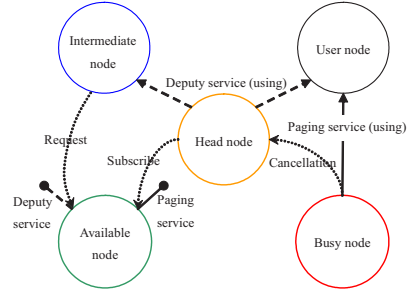
A RAM Grid preserves the paging mechanism of traditional operating systems. This means that the basic unit of the system is a “page” and the basic operations are “put page” and “get page”. We adopt the service-oriented view in DPRG, which allows the grid memory resources to publish their services, and the requestor to transparently discover these services and access grid memory.

There are five sets of nodes and two types of services in DPRG. The classification and relationships of the nodes and services are illustrated in Figure 2. We explain them in detail in the following sections.

#### 3.1. Paging Service and Deputy Service

DPRG provides two basic services: paging service and deputy service, based on the Open Grid Service Architecture (OSGA)[8, 10]. The flexibility of services provides open and uniform interfaces for heterogeneous resources. Note that current Web Service protocols can not be used directly due to their poor performance. We will study and design proper protocols for DPRG in future works.

A node providing *paging service* allows other nodes in the system to remotely store a page in its shared



**Figure 2. Relationship of different nodes and services.**

memory and fetch the page at any time. A node providing *deputy service* acts as an agent that assists the service requestor by discovering paging services. The service requestor does not search for paging services directly; it relies on the deputy service to do it. The deputy service greatly releases the service requestor from resource preparation, since the latter is already heavily loaded with scientific computing tasks.

#### 3.2. Node Sets and Node States

DPRG is comprised of five sets of nodes:  $S_{\text{available}}$ ,  $S_{\text{busy}}$ ,  $S_{\text{intermediate}}$ ,  $S_{\text{user}}$ , and  $S_{\text{head}}$ . The relationship between them is as follows:

$$\begin{aligned} \bigcup_i S_i &= S \\ \forall S_i \cap S_j &= \Phi \quad i \neq j \end{aligned} \quad (3)$$

where  $S$  is the set of all the nodes in the system. The five node sets correspond to the five states. If node  $N$  belongs to set  $S_x$  ( $N \in S_x$ ), it means “ $N$  is at state  $X$  and it is an  $X$  node”. Therefore, the expressions “ $N \in S_x$ ”, “ $N$  is at state  $X$ ” and “ $N$  is an  $X$  node” are equivalent statements in this paper. A node’s state tells us the current behavior of the node.

**Available node:** If a node has a certain amount of free memory to provide a paging service, it is an available node. Each node monitors its own memory utilization and makes the decision about its own availability by the *paging availability condition*. A small portion of the available nodes that satisfy the *deputy availability condition* can provide deputy service.

**Busy node:** If a node’s paging service is being used by another node, it is a busy node. A busy node constantly monitors its own performance and decides whether or not to continue providing service.

**Head node:** If a node is providing deputy service, it is a head node. Head nodes are selected from available nodes by the *deputy availability condition* which emphasizes free CPU cycles over free memory.

**User node:** If a node is using one or more paging services provided by busy nodes or a deputy service provided by a head node, it is a user node.

**Intermediate node:** If a node is not available to provide any paging or deputy services, and is not using any paging services, it is at an intermediate state. An intermediate node consumes a large portion of the local memory, but it is unclear whether it will turn into an available state or a user state in the near future.

Node states are related to the nodes' actions in the two services. The available, busy, and head nodes are the service providers in different situations. The user and intermediate nodes are the service consumers. Each user node or intermediate node is associated with one head node.

State transitions are triggered by certain service operations, which we call events. Figure 3 depicts the state diagram including seven transitions.

There are some consecutive and concurrent transitions in the above diagram.

I. When an available node becomes an intermediate node ①, it requests a deputy service to discover paging services for later usage, so another available node must be quickly selected as the head node to run this deputy service ⑤.

II. When an intermediate node becomes available ③, the deputy service it is using should be released, and the corresponding head node also becomes available ⑥.

III. When an intermediate node becomes a user node ②, at least one available node becomes busy and provides paging services for it ⑤.

IV. When the user node returns back to intermediate state ④, it releases all of its paging services, and the corresponding busy nodes becomes available ⑥.

### 3.3. Paging Service Operations

As previous discussed, paging service is the basis for providing grid memory, including the core operations *put page* and *get page*, and the other management operations *service release*, *service withdrawal*, and *service subscription*.

#### 3.3.1. Put Page and Get Page

As illustrated in Figure 4, the two basic paging operations are *put page* and *get page*. *Put page* stores a page from the user node to the busy node. *Get page* fetches a page from the busy node to the user node. The user node manages a page table for all remote pages so that it can determine which remote page to use for *put page* and where to use *get page*.

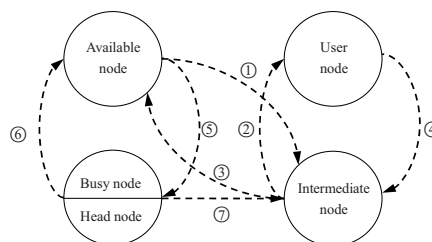


Figure 3. Node state diagram.

#### 3.3.2. Paging Service Subscription

The paging service subscription operation is used to inform an available node that its service may be used in the near future by a head node. A successful subscription adds a record into the subscription list of the head node. After that, the service can be used at any time until a cancellation message is received. Each available node maintains a subscriber list. When the available node changes states, it should send a cancellation message to all the head nodes on its subscriber list.

#### 3.3.3. Paging Service Release

Each paging service is associated with a timer to record the amount of time elapsed since the last *put page* or *get page* operation. If a paging service does not store any pages of the user node and has not been used for  $T_{idle}$ , it is implicitly released, as illustrated in Figure 5.

The paging service then informs the user node and the head node about the service release. The user node stores any shortcuts of the paging services that it has ever used. The next time the user node needs memory services, it will first request these shortcut services.

#### 3.3.4. Paging Service Withdrawal

Paging service withdrawal stops a busy node from providing service for a user node in the middle of the service process according to the *paging unavailability condition*. The busy node is responsible for handing over its jobs to another available node. To ensure a quick service handover, each busy node subscribes to some available backup services. As illustrated in

Figure 6, the process of the service handover proceeds as follows:

- The busy node swaps some of the user's page frames into the hard disk in order to satisfy the local memory demand with higher priority.
- It asks its backup nodes about their service parameters. It then selects an appropriate backup node as the substitute and moves the user's page frames to the substitute.

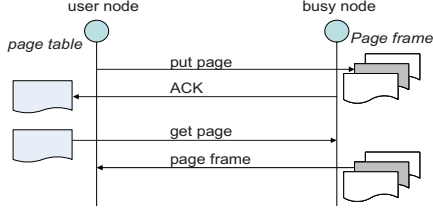


Figure 4. Put page and get page operation.

- c. It informs the head node about its handover to update the usability list.
- d. It informs the user node about its handover to update the page table.
- e. The busy node turns into an intermediate node.

### 3.3.5. Paging Service Monitoring

We first define  $U_{RAM}$  ( $0 < U_{RAM} < 1$ ) as the availability threshold. Let  $P_{RAM}$  denote the predicted memory utilization ratio of node  $G$  at time  $t$ . Node  $G$  is considered to be available to provide a paging service at time  $t$  if and only if  $P_{RAM} < U_{RAM}$ . This is the *paging availability condition*.

The paging service capacity  $M_{share}$  of an available node is defined as the maximum number of pages that an available node can share. Let  $M_{total}$  denote the total number of page frames of  $G$ , and  $M_{local}$  denote the number of page frames being used by  $G$ 's local application. We introduce  $M_{upper}$  to represent the upper bound of  $M_{share}$ .

$$M_{upper} = M_{total} - \max\{r \times M_{local}, M_{total} \times U_{RAM}\}$$

or

$$M_{upper} = M_{total} - r \times M_{local} \quad (4)$$

Here we give two different calculations of  $M_{upper}$ . The former is stricter, while the latter is looser. Here  $r$  is called the relax factor (usually  $r = 2$ ).  $M_{share}$  can then be calculated as follows.

#### The initial value of $M_{share}$

In the duration of a paging service process,  $M_{share}$  does not change. At the end of each service process (service release or withdrawal), the value of  $M_{share}$  is updated and the new  $M_{share}$  is applied to the next round of paging services. The initial value of  $M_{share}$  is:

$$M_{share} = M_{upper} \quad (5)$$

This value of  $M_{share}$  is used when an available node joins the system.

#### First tier update of $M_{share}$

At the end of a paging service process, if the service is successfully released, we should increase  $M_{share}$  by the proper amount:

$$M_{share} = M_{share} + (M_{upper} - M_{share}) / 2$$

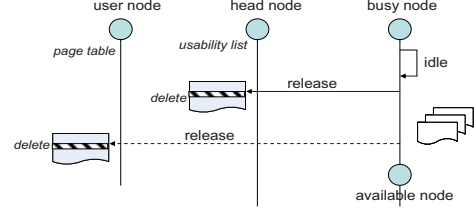


Figure 5 Paging service release operation.

$$= (M_{upper} + M_{share}) / 2 \quad (6)$$

where  $M_{upper}$  is the current upper bound calculated from formula (4).

#### Second tier update of $M_{share}$

If a busy node  $G$  withdraws its paging services, it changes from a busy state to an intermediate state. To avoid unnecessary service withdrawal overheads in the future, we decrease  $M_{share}$ :

$$M_{share} = M_{share} / 2 \quad (7)$$

We can also define the *paging unavailable condition* here. Let  $M_{total}$ ,  $M_{local}$  and  $M_{share}$  be the current total, locally used, and shared page frames of a busy node at time  $t$ , respectively. A busy node is considered to be unavailable to provide paging services at time  $t$  if and only if  $M_{local} > (M_{total} - M_{share})$ .

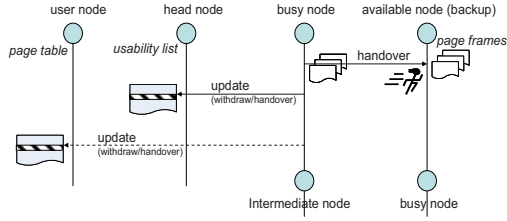
## 3.4. Deputy Service Operations

DPRG requires each intermediate and user node to be connected to a head node to provide deputy services. The intermediate node first searches for a number of available nodes using the service discovery approach described in Section 3.3, and then it selects the head node that has the highest processing power and acceptable network conditions. The user node inherits the deputy services of the intermediate node from which it is transformed.

The deputy service is an agent between the user and its paging services. Besides the *put page* and *get page* operations, other paging service management operations must go through the head node. The head node should maintain two lists for the user, a subscription list and a usability list.

### 3.4.1. Deputy Service Process

The deputy service maintains a list of subscribed paging services as follows. The deputy service starts at the time an available node turns into a head node. The head node searches available nodes by flooding or other existing search techniques. It then selects a set of appropriate paging services, whose aggregate subscription capacity is given by:



**Figure 6. Paging service withdrawal operations.**

$$\sum M_{share} \approx M_U \times b \quad (8)$$

where  $M_U$  is the local memory size of the user,  $b$  is the backup factor (usually  $b = 1.5$ ), and  $\sum M_{share}$  is the subscription capacity. The purpose of the backup factor  $b$  is to over-subscribe a certain amount of service capacity in case some services are cancelled later. The head node then subscribes to the selected available paging services and maintains a subscription list.

When the user node has a page P that needs to be stored in grid memory, but the capacity of its current paging services are used up, the following steps are carried out, as illustrated in Figure 7.

- The page P is sent to the corresponding head node.
- The head node immediately sends the subscription list to the user node.
- The head node selects a subscribed paging service S to store P.
- The head node informs the user node of its selection of S.
- If the head node receives more page frames from the user node, steps a, c and d are repeated.
- The user node receives the subscription list, and uses the paging service directly.

After a head node sends the subscription list to the user node, it starts a new round of discovery and subscription. The discovery and selection approach is the same as that in step a). However, this time the aggregated subscription capacity is given by:

$$\sum M_{share} \approx (M_R + \sum M_P) \times b \quad (9)$$

where  $\sum M_P$  is the total capacity of the paging services being used by R. This update of subscription capacity conforms to the “exponential request” criteria of Section 2. Step b) and c) are repeated during the process of the deputy service.

### 3.4.2. Settlement of Subscription Cancellation

DPRG requires the subscribed paging service to send a cancellation message to all (head) nodes in the

subscriber list. As time elapses, more paging services are cancelled and the head node’s subscription capacity is gradually reduced, as illustrated in Figure 8. To maintain a subscription list of a certain capacity, the head node needs to conduct re-discovery and re-subscription. Due to the overhead of service discovery, DPRG only conducts re-discovery and re-subscription when the following condition is satisfied:

$$\sum M_{share} < (M_R + \sum M_P) \quad (10)$$

Notice that the above formula does not use the backup factor  $b$ . This means that re-discover and re-subscription is conducted only when the subscription capacity is not sufficient. The re-subscription capacity is the same as in formula (8) and (9).

### 3.4.3. Deputy Service Release

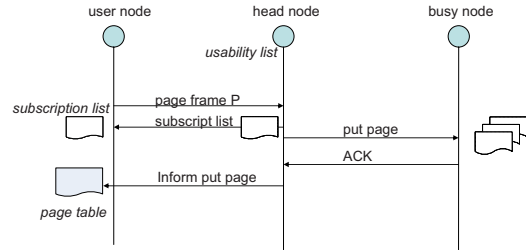
When all the paging services are released, the user node enters the available state. Its deputy services and all the subscribed paging services in the subscription list are released, and the head node also turns into an available node, as illustrated in Figure 9.

### 3.4.4. Deputy Service Withdrawal

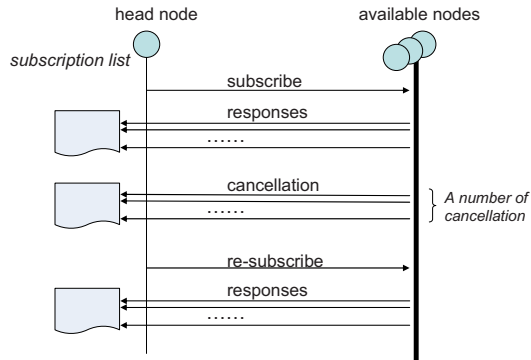
A head node withdraws the deputy service when it is determined to be unavailable according to the *deputy unavailability condition*. It hands over the subscription list and usability list to a new head node and then informs the user and all available nodes in the subscription list (and all busy nodes in the usability list) about the head node transfer.

### 3.4.5. Deputy Service Monitoring

Unlike paging services, deputy services do not have much of a memory requirement. Instead, deputy services need more processing power for service maintenance. At time  $t$ , let the predicted CPU utilization ratio of the node N in the next  $T$  seconds be  $P_{CPU}$ .



**Figure 7. Operations of deputy service process.**



**Figure 8. Settlement of paging service cancellation.**

We select the node that has the lowest value of  $P_{CPU}$  to be the head node. This is the deputy *availability condition*.

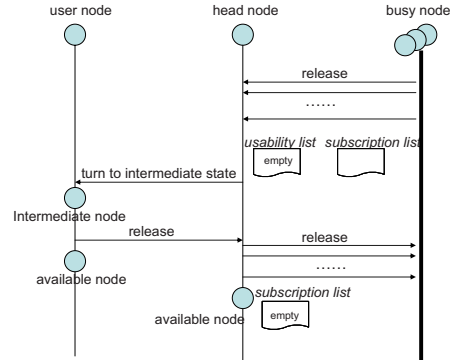
We also define the *deputy unavailability condition* as follows. A head node H is considered to be unavailable to provide deputy services any further if and only if its CPU utilization in the past T seconds,  $P_{cpu}$ , is higher than a threshold  $U_{cpu}$ , i.e.  $P_{cpu} > U_{cpu}$ .

If the *deputy unavailability condition* is satisfied, the head node withdraws its ongoing deputy service and hands it over to a backup node selected from the subscription list. The detailed process is described in Section 3.4.4.

#### 4. Performance Evaluation

We collect trace from a real grid system, SkyHawk. The SkyHawk grid is currently composed of the data grid system GridDaen, the database grid system GridDaen-EDS, and the grid monitoring system GridEye. Both GridDaen and GridEye are important modules integrated into the China National Grid system, which is the largest test-bed for grid technologies in China, as illustrated in Figure 10. Several large-scale applications in many industries, including National Meteorological Grid, Spatial Information Grid, and Remote Sensing Data Processing system are currently deployed in SkyHawk.

In order to simulate the memory requirements of real applications, we modified the system kernel to record the page swap actions between the local memory and disk when workstations are running the meteorological analysis application. The trace information includes page swap time (in microseconds), swap type (swap in or swap out), and the page location in a disk. In SkyHawk, the standard setting is a Linux workstation with 2 Intel Xeon 3.6GHz processors and at most a 4G physical memory. We modify and install the Linux kernel version 2.4.22 on RedHat release 9 to record trace information. Once we

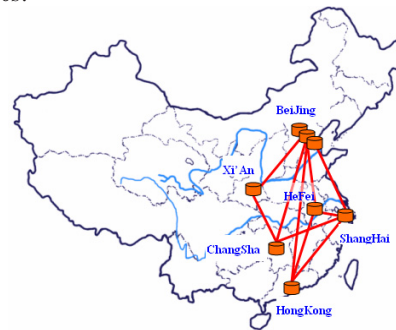


**Figure 9. Deputy service release operation.**

have the data, we launch a trace-driven simulation to evaluate the performance of DPRG.

#### 4.1. DPRG vs. Disks

Our first concern with the RAM Grid is the network conditions, which might restrict the grid memory's access speed. Without DPRG, the replaced page will be swapped to hard disks. We compare the completion time using DPRG and disks under different network conditions, changing the average WAN bandwidth and latency. In both tests, the completion time of disks is always the same, in spite of the change in network condition. In Figure 11, when the bandwidth increases from 1M to 16M, the completion time decreases slowly due to the faster remote page transmission. In Figure 12, the completion time increases with the end-to-end latency. We test five latencies, 1/4T, 1/2T, T, 2T, and 4T, where T is the default average latency (1.5ms). In both figures, DPRG's completion time is much lower than that of disks. The former is always less than 150s no matter what the network condition is, while the latter is more than 300s.



**Figure 10. The SkyHawk grid.**

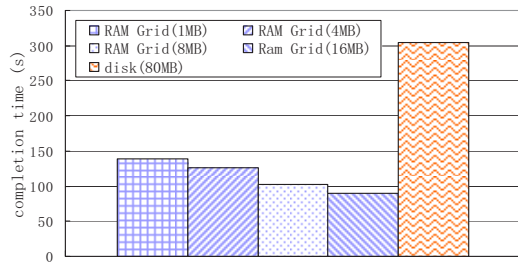


Figure 11. RAM Grid V.S. disk under different network bandwidth (1M – 16M). The disk bandwidth is 80M.

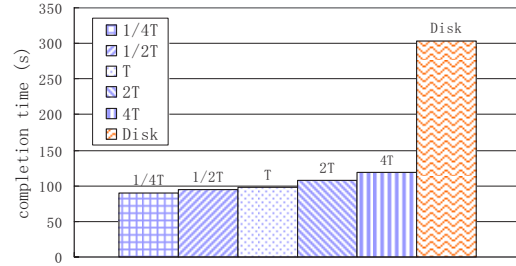


Figure 12. RAM Grid V.S. disk under different network latency (1/4T – 4T).

#### 4.2. RAM Grid vs. Network Memory

Network memory, without any mechanisms for heterogeneous and dynamic Internet environments, cannot be directly used in grid systems. In this experiment, we compare DPRG with network memory schemes.

We study two parameters: the heterogeneous factor and the dynamic factor, and their effect on RAM Grid and network memory. The heterogeneous factor ( $F_h \in [0,1]$ ) denotes the heterogeneity (including architecture, code set, operating system, etc.) of grid nodes.  $F_h=0$  implies that all grid nodes are homogenous; and  $F_h=1$  implies that all grid nodes are heterogeneous. Figure 13 plots the completion time of DPRG and network memory under different values of  $F_h$ . Since RAM Grid can utilize all heterogeneous resources, its performance does not change with  $F_h$ . Network memory, however, increases exponentially. This means network memory is severely affected by the heterogeneous factor. The dynamic factor ( $F_d \in [0\%,100\%]$ ) indicates the possibility of unexpected service failure while the service is being used. The RAM Grid can deal with these dynamic conditions using backup nodes and the overhead is overlapped by the head node, and thus, it is not affected by the dynamic factor. Network memory, however, is sensitive to service dynamicity. In Figure 14, we can see that the completion time of network memory increases linearly with the dynamic factor.

#### 4.3. Different System Workload

We evaluate RAM Grid under different system workloads. The system workload is changed by either adjusting the application interval (the time interval between the termination of a scientific application and the start of the next application), or by adjusting the

proportion of intermediate nodes, slight-busy nodes, and heavy-busy nodes.

In Figure 15, when the application interval increases, the completion time of the application becomes shorter. We test different application intervals from 1-15min to 40-480min. The decrease is almost flat when the distribution of the interval is in a range higher than 5-60mins in Figure 15. This means that at this point, RAM Grid is already able to provide sufficient grid memory. The disk performance is also plotted in the figure as a reference. We also test different node proportions and observe that the completion time increases when there are more heavy-busy nodes. From Figure 16, we can see that the completion time increase of DPRG is much slower than that of the disk.

#### 4.4. Different Parallel Size

In our previous experiments, we investigated the impact of parallel jobs of size  $F_s$ .  $F_s$  denotes the number of cluster nodes participating in one parallel job.  $F_s = 1$  means that only one cluster node is participating in the computing, so synchronization is not required.  $F_s = 32$  means all of the 32 nodes in a cluster are joined in the computing, such that the synchronization overhead will be high. In Figure 17, we can see that when the parallel job size increases, the completion time of RAM Grid increases at a much slower pace than that of the disk. This means that RAM Grid is not as affected by the cluster’s parallel job size.

### 5. Related Work

The page fault problem in parallel computing has been recognized and studied intensively in past years. Many solutions have been proposed; some focus on reducing the page fault overhead, while others try to reduce the number of the page faults itself.

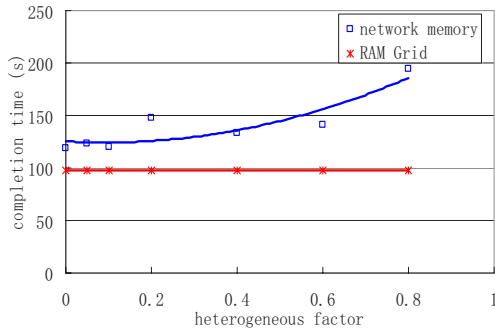


Figure 13. RAM Grid V.S. network memory when the heterogeneous factor is changing.

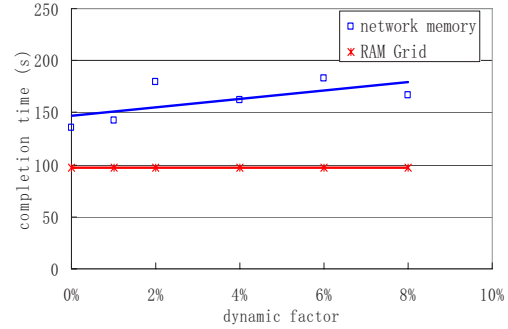


Figure 14. RAM Grid V.S. network memory when the dynamic factor is changing.

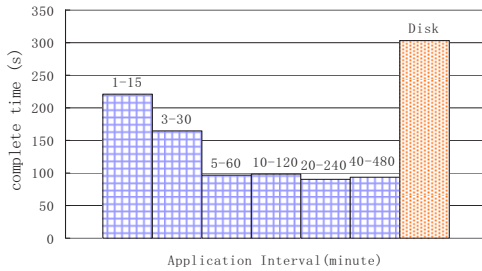


Figure 15. The impact of the application interval on RAM Grid performance.

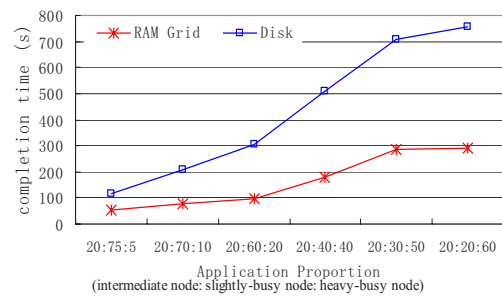


Figure 16. The impact of the application proportion on RAM Grid performance.

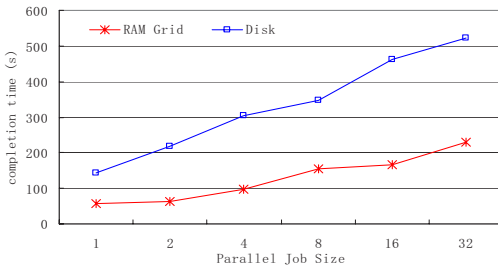


Figure 17. The impact of the parallel job size on RAM Grid performance.

Many variations of the scheduling algorithm have been proposed to prevent the occurrence of page faults. Douglas C. Burger *et al* point out that in parallel computing, CPUs, as well as memories must be “gang scheduled” [1]. Later, Anat Batat *et al* propose an improved gang scheduling algorithm that takes memory into consideration [11]. They propose that instead of scheduling a task to the node that cannot fulfill its memory requirements, it is better to suspend the task and wait for another appropriate node. This is a multi-criteria

scheduling problem known to be NP-hard. Anat Batat *et al* use the execution history and other information extracted from execution files in his prediction. This method, however, is not suitable in the case of dynamic memory allocation, such as heap memory.

Two kinds of approaches are usually used to reduce the page fault overhead. The hard-approaches use high-speed paging devices. They provide faster paging than a traditional hard disk. The soft-approach uses network memory as the secondary RAM. Michael J. Feeley *et al* build GMS (Global Memory Service), and the global memory management system used in cluster computing. They make a comprehensive design of the overall system that considers many problems encountered in network memory and developed solutions such as the page replacement algorithm, the reliability problem, and so on [2, 12, 13]. They adopt a distributed directory in page management, where each page has a globally unique ID. The whole system was implemented at the low operating system layer. Michael J. Feeley *et al* analyze the workload of the memory server and improve the GMS performance by using methods of sub paging and pre-fetching [12, 14]. Anurag Acharya *et al* analyze

parameters concerning idle memory (idle probability, the idle interval length, etc.) from a large amount of trace data [15]. They then build a run-time system called Dodo for harvesting idle memory. Evangelos P. Markatos *et al*'s research focuses on reliability. They build a remote memory pager similar to GMS. By creating a parity server for remote page checking, the system's usability can be increased [16].

All the above network memory approaches are designed for a single cluster. John Oleszkiewicz *et al* point out that the performance of network memory is sensitive to the workload and network conditions of the whole cluster [3]. In situations where some cluster nodes are under a heavy load, or a portion of the cluster network is congested, the network memory performance drops rapidly. However, this balance is pretty limited since the scope of a cluster is fairly small. If some nodes fail to provide enough memory, or some paths are congested, it is probable that other nodes also have heavy jobs and other paths will be influenced by the congested link.

## 6. Conclusion

The major contributions of this paper are as follows.

- We propose RAM Grid, a grid memory sharing mechanism. We further define a series of design criteria, such as single service offer, exponential request, and non-dual-identity, which are applicable to RAM Grid systems.
- We design the DPRG scheme, which is composed of two major services: paging service and deputy service. We provide a detailed service design, including most of the core functions needed in grid memory sharing.
- Through trace-driven simulations, we evaluate the performance of DPRG and compare it with the network memory in grid environments. Experimental results show that DPRG significantly outperforms existing remote memory sharing schemes and effectively supports grid computing applications.

The deployment of DPRG in Skyhawk Grid is currently ongoing in our lab.

## References

- [1]D. C. Burger, R. S. Hyder, B. P. Miller, and D. A. Wood, "Paging tradeoffs in distributed-shared-memory multiprocessors," In Proceedings of Conference on High Performance Networking and Computing, Washington, D.C., 1994.
- [2]M. J. Feeley, W. E. Morgan, F. H. Pighin, A. R. Karlin, H. M. Levy, and C. A. Thekkath, "Implementing Global Memory Management in a Workstation Cluster," In Proceedings of Symposium on Operating Systems Principles, Copper Mountain Resort, Colorado, 1995.
- [3]J. Oleszkiewicz, L. Xiao, and Y. Liu, "Parallel Network RAM: Effectively Utilizing Global Cluster Memory for Large Data-Intensive Parallel Programs," In Proceedings of the 2004 International Conference on Parallel Processing, Montreal, Quebec, Canada, 2004.
- [4]M. Wu and X.-H. Sun, "Memory Conscious Task Partition and Scheduling in Grid Environments," In Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing, Pittsburgh, 2004.
- [5]H. Sun, L. Zhong, J. Huai, and Y. Liu, "OpenSPACE: An Open Service Provisioning and Consuming Environment for Grid Computing," In Proceedings of First IEEE International Conference on e-Science and Grid Computing, Melbourne, Australia, 2005.
- [6]Y. Liu, X. Liu, L. Xiao, L. M. Ni, and X. Zhang, "Location-Aware Topology Matching in P2P Systems," In Proceedings of IEEE INFOCOM 2004, Hong Kong, China, 2004.
- [7]I. Foster and A. Iamnitchi, "On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing," In Proceedings of 2nd International Workshop on Peer-to-Peer Systems Berkeley, 2003.
- [8]I. Foster, K. Czajkowski, and D. Ferguson, "Modeling and Managing State in Distributed Systems: The Role of OGS and WSRF," In Proceedings of the IEEE, 2005.
- [9]D. A. Patterson, "Latency lags bandwidth," *Communications of the ACM*, vol. 47, pp. 71-75, 2004.
- [10]I. Foster, C. Kesselman, J. Nick, and S. Tuecke, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration.," In Proceedings of the 5th Global Grid Forum Workshop (GGF5), 2002.
- [11]A. Batat and D. G. Feitelson, "Gang Scheduling with Memory Considerations," In Proceedings of the 14th International Symposium on Parallel and Distributed Processing, Cancun, Mexico, 2000.
- [12]H. A. Jamrozik, M. J. Feeley, G. M. Voelker, J. E. II, A. R. Karlin, H. M. Levy, and M. K. Vernon, "Reducing Network Latency Using Subpages in a Global Memory Environment," In Proceedings of Seventh International Conference on Architectural Support for Programming Languages and Operating Systems, Cambridge, Massachusetts, 1996.
- [13]G. M. Voelker, H. A. Jamrozik, M. K. Vernon, H. M. Levy, and E. D. Lazowska, "Managing server load in global memory systems," In Proceedings of ACM SIGMETRICS International Conference on Measurements and Modeling of Computer Systems, Seattle, Washington, United States, 1997.
- [14]G. M. Voelker, E. J. Anderson, T. Kimbrel, M. J. Feeley, J. S. Chase, A. R. Karlin, and H. M. Levy, "Implementing cooperative prefetching and caching in a globally-managed memory system," In Proceedings of Joint International Conference on Measurement and Modeling of Computer Systems, Madison, Wisconsin, United States, 1998.
- [15]A. Acharya and S. Setia, "The Utility of Exploiting Idle Memory for Data-Intensive Computations," *Technical Report: TRCS98-02*, 1998.
- [16]E. P. Markatos and G. Dramitinos, "Implementation of a Reliable Remote Memory Pager," In Proceedings of USENIX Annual Technical Conference, San Diego, CA, 1996.