PAPER

# Grid Resource Monitoring and Selection for Rapid Turnaround Applications

**Kensuke MURAKI**[†*], *Nonmember*, **Yasuhiro KAWASAKI**[†], *Student Member*, **Yasuharu MIZUTANI**[††], **Fumihiko INO**[†a)], *and* **Kenichi HAGIHARA**[†], *Members*

**SUMMARY** In this paper, we present a resource monitoring and selection method for rapid turnaround grid applications (for example, within 10 seconds). The novelty of our method is the distributed evaluation of resources for rapidly selecting the appropriate idle resources. We integrate our method with a widely used resource management system, namely the Monitoring and Discovery System 2 (MDS2), and compare our method with the original MDS2 in terms of the performance and the scalability. The performance is measured using a 64-node cluster of PCs and the scalability is analyzed using a theoretical model and the measured performance. The experimental results show that our method reduces the resource selection time by 82%, as compared with the original MDS2. The scalability analysis also indicates that our method can keep the resource selection time within 1 second, up to 500 nodes in local-area-network (LAN) environments. In addition, some simulation results are presented to estimate the impact of our method for wide-area-network (WAN) environments.

***key words:*** *grid computing, resource management, rapid turnaround, evaluation*

## 1. Introduction

Grid technology [1], [2] plays an increasingly important role in building a high-performance computing (HPC) environment in a virtual organization. This technology allows us to share various computing resources distributed in different organizations, providing us a coordinated HPC environment. It realizes various grid-enabled computations, such as acceleration of compute-intensive applications using idle computing resources in home and/or office [3]–[5] and that of data-intensive applications using storage resources [6]–[8]. Note here that we follow Ian Foster's definition [2] of the Grid, so that the Grid in the present paper consists of local-area-network (LAN) and/or wide-area network (WAN) connected resources.

In order to minimize the effort for developing such grid-enabled systems, many research projects [5], [9]–[11] provide useful toolkits and frameworks. For example, the Globus Toolkit 2 (GT2) [9] is a standard middleware used for building such systems. These prior projects successfully provide high performance, high throughput, secure computing environments for large-scale applications running for days or weeks.

Thus, prior projects mainly focus on accelerating such long-term applications. In this situation, resource monitoring and selection systems are allowed to take a few ten seconds (or even minutes) to determine which resources to be used for the submitted job. Actually, it takes about 30 seconds to aggregate resource information from 500 nodes in a LAN environment [12]. Although this overhead is small enough for long-term applications, it is a critical problem for short-term applications which require their computation results in rapid turnaround time, for example, within 10 seconds. Thus, to execute short-term applications using many resources on Grids, we have to minimize the overhead for resource monitoring and selection. Note here that Kondo et al. [13] also focus on rapid turnaround applications. However, from our viewpoint, their focus is long-term applications which take approximately tens of minutes, so that the overhead mentioned above is small enough.

In contrast to these prior projects, we aim at providing virtual HPC environments also for short-term applications. For example, grid technology would become more useful if it could reduce turnaround time for time-consuming medical applications using a virtual HPC machine integrated from idle PCs in a hospital [14]. In this case, reducing execution time from hours (on a single CPU system) to seconds makes it possible to innovate in novel computer-assisted surgery for increased surgical safety and accuracy. Thus, as Ian Foster says in [2], grid technology should realize a coordinated system capable of appropriately selecting resources for users to satisfy their various time constraints. From this viewpoint, short-term applications should be allocated to LAN-connected (ambient) resources due to lower network latency. On the other hand, long-term applications that do not have strict time constraints might be allocated to WAN-connected resources with higher computational performance.

In this paper, we propose a novel method capable of selecting idle resources rapid enough for short-term grid applications. To reduce the overhead for resource selection, our method uses a distributed evaluation strategy that reduces the amount of data communicated and aggregated by the resource selection server and also reduces the workload of the server. Furthermore, our method is implemented using MDS2 [15], a resource monitoring system distributed with the standard GT2 and used in many grid-enabled sys-

Manuscript received November 7, 2005.
Manuscript revised March 17, 2006.

[†]The authors are with the Graduate School of Information Science and Technology, Osaka University, Toyonaka-shi, 560–8531 Japan.
[††]The author is with the Faculty of Information Science and Technology, Osaka Institute of Technology, Hirakata-shi, 573–0196 Japan.
[*]Presently, with the Nintendo Co., Ltd.
a) E-mail: ino@ist.osaka-u.ac.jp

DOI: 10.1093/ietisy/e89–d.9.2491

tems [16]–[18]. This compatibility with a standard middleware allows us to collect more resources from various organizations, because grid systems should use standard, open, general-purpose protocols and interfaces [19]. This also motivates us to exploit Grids over multiple organizations rather than clusters [20] in a single organization.

The remainder of the paper is as follows. Section 2 presents an overview of MDS2. Section 3 describes our proposed method, and then Section 4 shows some experimental results. Section 5 introduces related works. Finally, Sect. 6 concludes the paper.

## 2. MDS2

In order to provide useful information to a grid resource broker, who selects and allocates grid resources for jobs submitted by users, MDS2 [15] manages static and dynamic information that represents the state of a Grid. For example, static information includes OS (operating system) version, CPU type, and total disk space, while dynamic information includes load average and available memory/disk space.

Figure 1 illustrates the MDS2 architecture. MDS2 has a hierarchical structure consisting of three components: Grid Index Information Service (GIIS), Grid Resource Information Service (GRIS), and Information Provider (IP). A GRIS runs on a computing node and manages IPs to gather the resource information of the node. An IP registers with a GRIS and behaves as a sensor that generates the information of a resource, such as OS version, load average, and so on. A GIIS acts as a server that aggregates resource information from registered GRISes in order to provide it to the resource broker.

MDS2 has a security framework capable of accepting/denying requests incoming to each component. For example, GRISes can authenticate incoming requests from different organizations, allowing us to restrict the access to specific information. Although security issues are important for grid middleware, we would like to focus on performance issues which we addressed in the paper.

### 2.1 Resource Monitoring

MDS monitors resources in the following three steps (see also Fig. 2 (a)).

S1. Information request: Whenever a GIIS receives an inquiry of resource information from a resource broker, who determines the nodes to be allocated for the inquiry, the GIIS requests resource information to registered GRISes. Then, each GRIS further requests resource information to registered IPs. Security process, namely authentication and access control is required between the GIIS and the GRIS.

S2. Information generation: Each IP probes its responsible resource so that generate resource information. The information generated by IPs are gathered and concatenated by the GRIS who manages generally several IPs.

S3. Information aggregation: The GIIS receives resource information from GRISes in order to aggregate them into global information, which represents the state of a Grid. As well as in step S1, the security process is also needed. Then, the aggregated information is provided to the broker.

Note here that the behavior mentioned above assumes no cache mechanism. The details of this mechanism are mentioned later in Sect. 2.2.

The overhead for information aggregation is defined as the time required to process steps S1–S3. In the following, we call this time the aggregation time.

### 2.2 Cache Mechanism

MDS2 has a cache mechanism in order to reduce the aggregation time. By using this mechanism, the GIIS and GRIS are allowed to omit the information request to lower-level components. Therefore, it reduces communication between components and prevents invoking lower-level components.

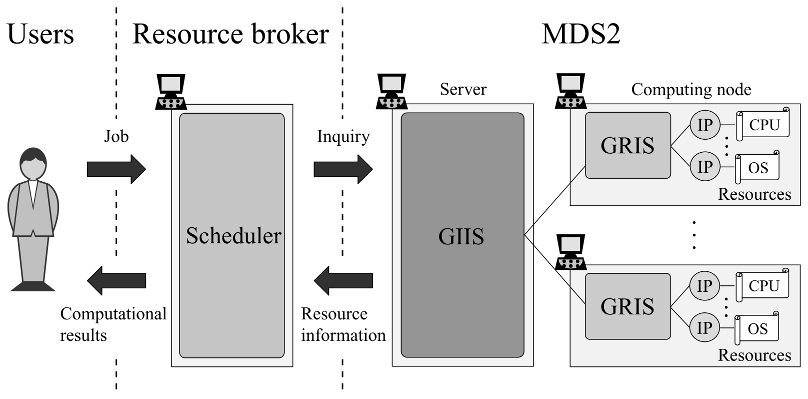


**Fig. 1** Architecture of MDS2. Resources are managed by a hierarchical structure consisting of three components: GIIS, GRIS, and IP have the information of a Grid, that of a computing node in the Grid, and that of a resource in the node, respectively.

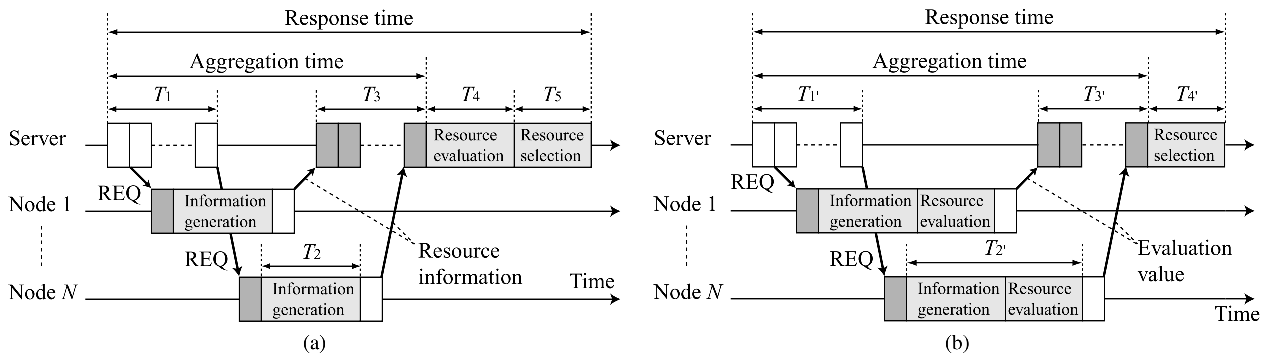


**Fig. 2** Timeline views of resource selection behavior using (a) original MDS2 and (b) our method. The cache mechanism is not used in this example. Our method differs from the original MDS2 in terms of (1) performing resource evaluation at computing nodes rather than the server and (2) communicating evaluation values in stead of resource information.

However, in stead of this omission, the cache mechanism requires a buffer in components to keep the last information they have received. This buffer allows us to immediately return the buffer data to higher-level components. In addition, the appropriate value is also needed for a cache TTL (time-to-live) parameter, which defines the expiration time of the cache data. This parameter is useful to prevent MDS2 from returning misleading information that does not represent the latest state of resources.

Thus, the cache mechanism is useful to achieve rapid turnaround processing on Grids. However, it can mislead the resource broker to an inappropriate selection of resources if the cache TTL parameter is inappropriately set to a large value.

### 2.3 Resource Selection

Because MDS2 is a resource monitoring system, it does not address the problem of resource selection. On the other hand, our method performs both resource monitoring and selection. Therefore, in order to compare both methods in terms of theoretical performance, we assume a simple selection model for MDS2. This model selects resources in the following two steps, in addition to steps S1–S3 (see Fig. 2 (a)).

S4. Resource evaluation: The broker receives the aggregated information from the GIIS in order to evaluate how computing nodes match the inquiry received in step S1. Here, the evaluation measure for each node is given by a function which requires the aggregated information and then returns an evaluation result in a numeric value representing the quality of the match between the resource and the requirements of the submitted job. We assume that this evaluation takes $O(N)$ time, where $N$ represents the number of computing nodes.

S5. Resource selection: Based on the evaluation results, the broker selects resources in order to allocate them to the inquiry. In our assumption, resource selection is done by a ranking mechanism that sorts the evaluation results to select the best match between the available resources and the requirements of the submitted job. Therefore, we assume that this step takes $O(N \log N)$ time. Note here that this complexity can be reduced to $O(N)$ time if top $n$ list is sufficient for resource selection, where $n \leq N$. Also, the insertion sort algorithm may be useful if steps S3, S4, and S5 can be processed at the same time.

The overhead for resource selection is defined as the time required to process steps S1–S5. In the following, we call this time the response time.

## 3. Proposed Method for Resource Monitoring and Selection

In this section, we describe our resource monitoring and selection method for rapid turnaround applications. Figure 2 (b) shows a timeline view for our method, presenting how we change the behavior of the original MDS2.

### 3.1 Distributed Evaluation Strategy

The key difference between our method and the original MDS2 is the distributed evaluation of resources, where computing nodes evaluate themselves to compute how they match the inquiry. Here, the evaluation measure is similar to the measure mentioned in Sect. 2.3, but it requires local information instead of global information. This distributed evaluation strategy contributes to achieve less communication and less workload at the resource selection server. After this evaluation, the server aggregates the evaluation results, namely values, from resources, and then selects resources to be allocated for the inquiry.

Summarizing the above description, our method has two advantages as follows.

- Less communication at the server. While the original MDS2 transmits the detailed resource information in a raw format, our method communicates only values. That is, the information of several resources in the

same node is summarized into a single value in order to reduce the amount of communication. This reduction allows us to achieve shorter response time. Furthermore, it also saves the network bandwidth for grid applications, allowing them to run faster with more bandwidth.
- Less workload at the server. The server has less workload, because computing nodes take the responsibility for resource evaluation. This distributed style of resource evaluation allows the server to concentrate on resource selection. In Fig. 2 (b), we can see that the resource evaluation step S4 in Fig. 2 (a) is parallelized using computing nodes in Fig. 2 (b). The resource selection can easily be done in $O(N \log N)$ time, as we mentioned in Sect. 2.3.

In contrast to the advantages mentioned above, there are some disadvantages.

- Lack of global, detailed state. Because our method communicates the evaluation values in stead of resource information, the server does not have the global, detailed resource information. Due to this lack, scheduling policies using this information is not available. Thus, resource evaluation must be represented as a local operation in our method. However, if each of the server and computing nodes has two MDS2 servers, one for our method and the other for the original MDS2, we can use such scheduling policies by choosing the original GIIS server for such policies.
- More workload at computing nodes. While the original MDS2 performs resource evaluation on the server, our method makes computing nodes evaluate themselves. Thus, our method slightly consumes computational resources in computing nodes. Although this evaluation overhead seems to be small, it might reveal itself as a problem if resource evaluation is frequently done. In this situation, resources could be wasted by our resource selection system rather than grid applications. However, this problem can be addressed by using the cache mechanism, which allows us to omit the resource evaluation step for a certain time.
- Additional communication for the evaluation function. Since computing nodes take the responsibility for resource evaluation, the evaluation function must be transmitted from the GIIS server to computing nodes. This overhead is not negligible compared to that for the evaluation results, namely the resource information. However, we think that typical functions are allowed to be cached or even stored in advance on the computing nodes to avoid this runtime transmission. In contrast, the evaluation results must be sent to the server for every inquiry, because they are time-dependent information. Thus, we think that the overhead for the evaluation function is not negligible, but a cache mechanism will avoid runtime transmission of the evaluation function.

```
evaluation_function() {
    if (os_name == íLinuxí && disk_space > 1GB
      && memory_space > 300MB)
        return cpu_clock_frequency *
          (1-load_average/100) + memory_space;
    else
        return 0;
}
```

**Fig. 3** Pseudo code showing an evaluation function.

## 3.2 Implementation Strategy Using MDS2

As we mentioned earlier, we implement our method on the MDS2 layer. The key point here is that our implementation strategy intends to collect more resources from different organizations by means of a standard middleware.

To achieve this, we develop an IP that does not only probe resources but also evaluate a computing node, according to an evaluation function. Furthermore, to avoid communicating raw resource information (generated by the original IPs) between the GIIS and GRISes, we unregister all the original IPs from GRISes, and then register only our custom IP. Our IP computes an integer value that represents how the computing node matches the submitted inquiry. For example, the function in Fig. 3 evaluates a computing node by means of local information, such as CPU speed, load average, and memory/disk space.

In summary, our implementation runs as follows.

S1'. Information request: This step is almost the same as in the original MDS2, but the request to the custom IP is sent to lower-level components.
S2'. Distributed resource evaluation: The newly added IPs generate information, namely evaluation values, in a distributed manner.
S3'. Information aggregation. See step S3 in Sect. 2.1.
S4'. Resource selection. See step S5 in Sect. 2.3.

The above strategy does not strictly implement our method, because it does not support runtime transmission of the evaluation function. According to [21], we expected this runtime transmission could be realized by passing the evaluation function with the inquiry to the custom IP, but this input interface seems not be implemented yet. An alternative solution is to develop a custom IP that downloads an executable file, namely the evaluation function, from a specific location of a specific GridFTP server [22], and then executes the downloaded file. This solution assumes that the evaluation function is uploaded to the GridFTP server in advance of the inquiry.

Note here that our implementation strategy allows the original MDS2 framework to work, because it requires only an additional registration of an IP. To do this (see [21] for details), we must (1) write a shell script as the custom IP, (2) distribute it to computing nodes, (3) register it at each node by modifying MDS2 configurations, and (4) restart the MDS2 service. Thus, this strategy requires modifications on computing nodes, which is not so flexible from a practical

**Table 1** Notations used in theoretical analysis.

| Notation | Explanation |
|---|---|
| $N$ | Number of nodes |
| $B$ | Network bandwidth between the GIIS server and computing nodes |
| $L$ | Network latency between the GIIS server and computing nodes |
| $S$ | Time for security process between the GIIS server and computing nodes |
| $A$ | Time for invoking IPs and gathering information in a computing node |
| $d$ | Data size of resource information that a computing node generate for a request |
| $f$ | Data size of the request or evaluation function transmitted from the GIIS server to computing nodes |
| $e$ | Time for evaluating a computing node using resource information |
| $s$ | Coefficient for the sorting operation |

viewpoint. However, the original MDS2 and our method can work together to provide resources for long-term and short-term applications, respectively. We have to utilize two MDS2 servers to do this.

## 3.3 Theoretical Performance Analysis

In this section, we construct a theoretical model to analyze the scalability of our method. We first model the response time for the original MDS2, and then that for our method. Table 1 shows the notations used in our model. Note here that the following model does not consider network congestion.

Let $T_i$ be the time spent for step S$i$, where $1 \leq i \leq 5$. Because the request message transmitted in step S1 is much smaller than the resource information in step S3, we assume that time $T_1$ is much shorter than time $T_3$. Thus, the amount of messages communicated by the server can be approximated by $dN$, where $d$ represents the amount of resource information that a computing node generates for the request. Then, each message needs to pass the security process. Therefore,

$$T_1 + T_3 = dN/B + 2L + 2SN, \tag{1}$$

where $S$ denotes the overhead for the security process for a message, and $B$ and $L$ represent the network bandwidth and the network latency between the GIIS server and GRISes, respectively. We also assume that step S2 takes $O(1)$ time, because this step is a simple local operation whose overhead depends only on the number of IPs:

$$T_2 = A, \tag{2}$$

where $A$ is the time for invoking IPs and gathering resource information from them. According to the assumptions mentioned in Sect. 2.3, the time for the remaining steps S4 and S5 is given by:

$$T_4 = eN, \tag{3}$$

$$T_5 = sN \log N, \tag{4}$$

where $e$ and $s$ represent the time for evaluating a computing node and the coefficient for the sorting operation, respectively.

Finally, the aggregation time can be approximated by $T_1 + T_3$ for the scalability analysis, namely for large $N$. That is, we can ignore time $T_2$, because $T_2$ is an $N$-independent cost associated with parallel step S3 while $T_1$ and $T_3$ increase with $N$. Thus, we assume that the scalability is limited by the GIIS server, which becomes busy in communicating with many computing nodes. Therefore, the response time for the original MDS2 without cache can be given as follows:

$$\begin{aligned} T_{MDS} &= T_1 + T_3 + T_4 + T_5 \\ &= (d/B + 2S + e + s \log N)N + 2L. \end{aligned} \tag{5}$$

In contrast, our method transmits only an evaluation value of 4 bytes from a node to the server. Also, it transmits an evaluation function from the server to nodes if it is not cached on the nodes. Therefore:

$$T_{1'} + T_{3'} = (f + 4)N/B + 2SN + 2L, \tag{6}$$

where $f$ represents the data size of the evaluation function. For step S2', we assume that it takes the same time as step S2: $T_{2'} = T_2$. Furthermore, the final step S4' is same as step S5: $T_{4'} = T_5$. Therefore, the response time for our method without cache is given by:

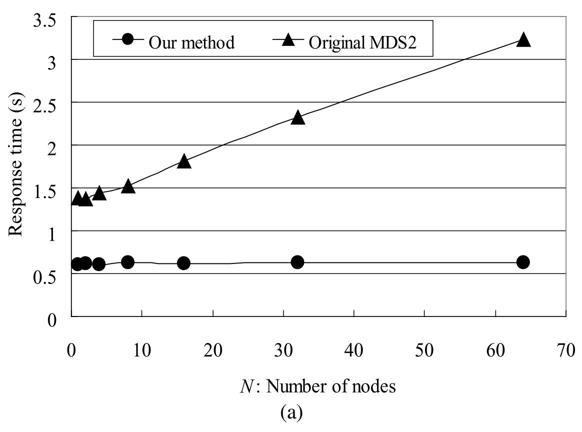
$$\begin{aligned} T_{prpsd} &= T_{1'} + T_{3'} + T_{4'} \\ &= ((f + 4)/B + 2S + s \log N)N + 2L. \end{aligned} \tag{7}$$

By comparing Eq. (5) with Eq. (7), we find that our model represents the two advantages mentioned in Sect. 3.1: (1) the communication time will be reduced from $dN/B+2L$ to $(f + 4)N/B + 2L$ (a) if the evaluation function is already sent to the nodes ($f$ is small) or (b) if the data size of the evaluation function is smaller than that of resource information ($f < d - 4$); and (2) term $eN$, namely the overhead for resource evaluation, is eliminated in our method.

Equations (5) and (7) also clearly indicate that WAN environments are not suited for rapid turnaround applications. Such networks usually have high latencies $L$ between the server and nodes. The latency $L$ must be at most 500 milliseconds in order to make the response time less than 1 second.

## 4. Experimental Results

We now show experimental results obtained on a cluster of PCs [20]. In the experiments, we evaluate our method from two viewpoints: the response time and the scalability. We also compare our method with the original MDS2.



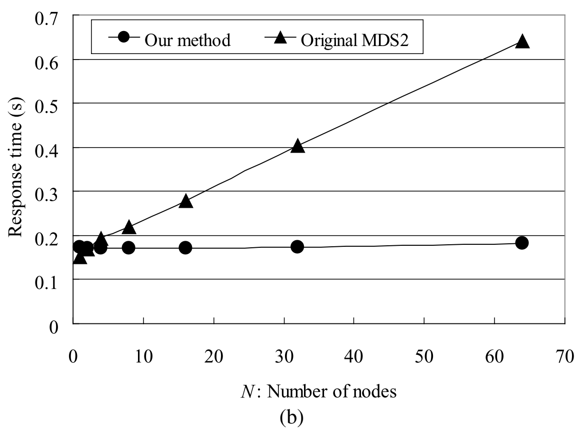


**Fig. 4** Measured response time (a) without cache and (b) with cache.

**Table 2** Breakdown of response time in seconds for 64 computing nodes. Times $T_1$ and $T_3$ are presented in summation, because steps S1 and S3 are processed simultaneously.

| Breakdown | w/o cache (s) | | w/ cache (s) | |
|---|---|---|---|---|
| | Proposed | MDS2 | Proposed | MDS2 |
| $T_1 + T_3$: Information request and aggregation | 0.10 | 2.70 | 0.17 | 0.58 |
| $T_2$: Information generation | 0.52 | 0.48 | — | — |
| $T_4$: Resource evaluation | — | 0.05 | — | 0.05 |
| $T_5$: Resource selection | 0.01 | | | |
| Response time (total) | 0.63 | 3.24 | 0.18 | 0.64 |

Our cluster has 64 computing nodes and one GIIS server, each equipped with two Pentium III CPUs running at 1000 MHz and 1133 MHz, respectively. All PCs have 2 GB of RAM and communicate each other through a 100 Mb/s LAN.

## 4.1 Response Time Analysis

We measure the response time for both methods using the evaluation function presented in Fig. 3. This function is implemented as a shell script with a file size of 898 bytes. Note here that the evaluation function is statically stored on computing nodes. Therefore, the following timing results are obtained under small $f$. These results include the overhead for security process. We also investigate the influence of the cache mechanism by running MDS2 with two configurations: the resource information data always in GIIS cache and the data never in cache. The former configuration is realized by setting the cache TTL parameter to a large value.

Figure 4 shows the response time measured with different numbers of nodes. The measured results indicate that (1) our method achieves shorter response time in most cases and (2) the performance gain to the original MDS2 increases with the number of nodes. For example, when using 64 nodes, our method reduces the response time by 72% for cache-disabled configuration and by 81% for cache-enabled configuration. Furthermore, the increase rate of the response time is also reduced from 29.4 (7.8) to 0.28 (0.14) ms/node for cache-disabled (cache-enabled) configuration, respectively.

To investigate this performance gain, we analyze the breakdown of the response time. Table 2 shows the breakdown for 64 nodes. Here, because our main focus is not the analysis of the original MDS2, we do not instrument the MDS2 code. Instead, we assume that the aggregation time can be approximated by $T_1 + T_2 + T_3$, because our cluster does not have enough nodes to make the server busy. Based on this assumption, we measured time $T_2$ on a node and estimated time $T_1 + T_3$ by subtracting $T_2$ from the aggregation time.

As we can see in Table 2, the reduction of time $T_1 + T_3$ mostly contributes to the performance gain. That is, our method reduces time $T_1 + T_3$ from 2.7 to 0.1 seconds by communicating values instead of raw resource information. Actually, the amount of data communicated by the server is reduced from 560 KB to 17.5 KB, so that the reduction rate of data size (1/32) is close to that of time (1/27).

Note here that the data size of 17.5 KB is the smallest value for 64 nodes. Although our method communicates only integer (4 bytes) values, MDS2 adds a header to each value, and this header increases the data size from 256 bytes ($= 4 \cdot 64$ bytes) to 17.5 KB.

In addition to the advantage of less communication, our distributed evaluation strategy also eliminates the resource evaluation overhead from the server. Although this overhead is not so large on our small cluster, it certainly reduces the response time by 0.05 seconds ($T_4$). Furthermore, this reduction is effective for cache-enabled configuration, which has shorter response time than cache-disabled configuration.

### 4.2 Scalability Analysis

We investigate the scalability of our method using the measured results and the theoretical model presented in Sect. 3.3.

According to the breakdown analysis presented in Table 2, time $T_4 + T_5$ accounts for less than 10% of the entire response time. Therefore, for large $N$, coefficients $e$ and $s$ in Eqs. (5) and (7) are small enough to approximate time $T_4 + T_5$ with zero. Thus, for large $N$, the response time $T_{MDS}$ and $T_{proposed}$ can be approximated by $(d/B + 2S)N$ and by $((f + 4)/B + 2S)N$, respectively.

By using this simplified model and the increase rate mentioned in Sect. 4.1, the response time for 500 nodes can be computed as 0.75 (0.24) seconds for our cache-disabled (cache-enabled) method and 16 (4) seconds for the original cache-disabled (cache-enabled) MDS2, respectively. Thus, even if we do not use the cache mechanism, our method will keep the response time within 1 second, up to 500 nodes in LAN environments. This estimated response time is rapid enough to run short-term applications on Grids. Moreover, resource selection will be carried out by the latest online information under cache-disabled configuration.

Note here that the analysis mentioned above assumes small $f$ due to the static distribution of the evaluation function. Since the runtime distribution increases $f$, the performance will decrease if the evaluation function is dynamically transmitted to computing nodes. In the experiments, the data size $d$ is reduced from 8 KB to 280 bytes. Therefore, the data size $f$ of the evaluation function must be less than 7912 bytes to obtain the performance gain of our method. It must also be mentioned that this requirement is for one-time evaluation functions. Therefore, there is no limitation on $f$ in a case of periodically updating resource information using the pretransmitted evaluation function.

### 4.3 Simulation Study

We now present some simulation results estimating the impact of our method on WAN environments. We use NS-2 [23], a network simulator, which provides precise simulation of packets traveling on the network. Because NS-2 simulates only the behavior of the network, we added dummy time to take count of the actual behavior of GRISes in the simulation. The actual time is measured using tcpdump to obtain the timestamp of incoming/outgoing packets with the packet size. Thus, we estimate the aggregation time for cache-disabled configuration.

Figure 5 shows the topology used for the simulation. We assume a simple topology in which the GIIS server and computing nodes are interconnected by a router. The router here has a drop-tail first-in, first-out (FIFO) queue with a size of 50 packets. Every network link has bandwidth $B$ in Mb/s and latency $L/2$ in seconds. The aggregation time is estimated for $N$ = 64, 128, 256, and 512 computing nodes.

Figure 6 shows the aggregation time estimated using

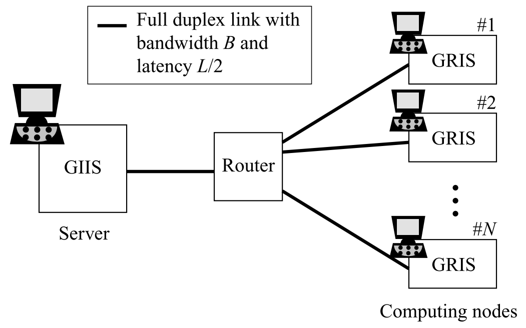


**Fig. 5** Simulation setup.

bandwidth parameter $B$ = 50 and different latency parameters $L$ ranging from 0.5 to 500 milliseconds. The estimated time for our method is less than 2 seconds if the latency is less than 50 milliseconds. In contrast, it takes more aggregation time on the original MDS2. This figure also shows that the latency of 500 milliseconds is too long for rapid turnaround applications.

We also measured the aggregation time with latency parameter $L$ = 0.05 and different bandwidth parameters $B$ (see Fig. 7). The results for the original MDS2 using $B$ = 1 cannot be obtained due to network congestion that causes serious packet loss. In such cases, simulations result in an error: REASON_SACK, hole fills in SACK [24]. According to these results, our method will achieve the aggregation time of less than 2 seconds on a 10 Mb/s network link up to 128 computing nodes. However, a 100 Mb/s link is required to monitor more than 128 nodes for rapid turnaround applications. In contrast, the original MDS2 takes approximately 10 seconds in most cases. Although we have failed to simulate its behavior on a 1 Mb/s network, it is obvious that the original MDS2 as well as our method will fail to provide successful time on such a high-latency, low-bandwidth network.

According to the simulation results mentioned above, one requirement for the evaluation function is that it must give a lower priority to high-latency resources. We think that such an evaluation function allows LAN-connected resources to execute only short-term jobs in a coordinated manner. Therefore, the entire Grid can satisfy various time constraints of user applications.

With regard to the practicality of our implementation strategy, the simulation results indicate that we can assume that resources with a higher latency clearly cannot achieve rapid turnaround computation. In other words, our custom IP should be distributed and activated only on low-latency nodes, for example, with a latency of 50 milliseconds. This restriction will improve the practicality of our method, because modifications are required only on promising nodes, namely a part of the entire Grid.

## 5. Related Work

Zhang et al. [12] study the performance of three representative monitoring systems: MDS2 [15] in the globus

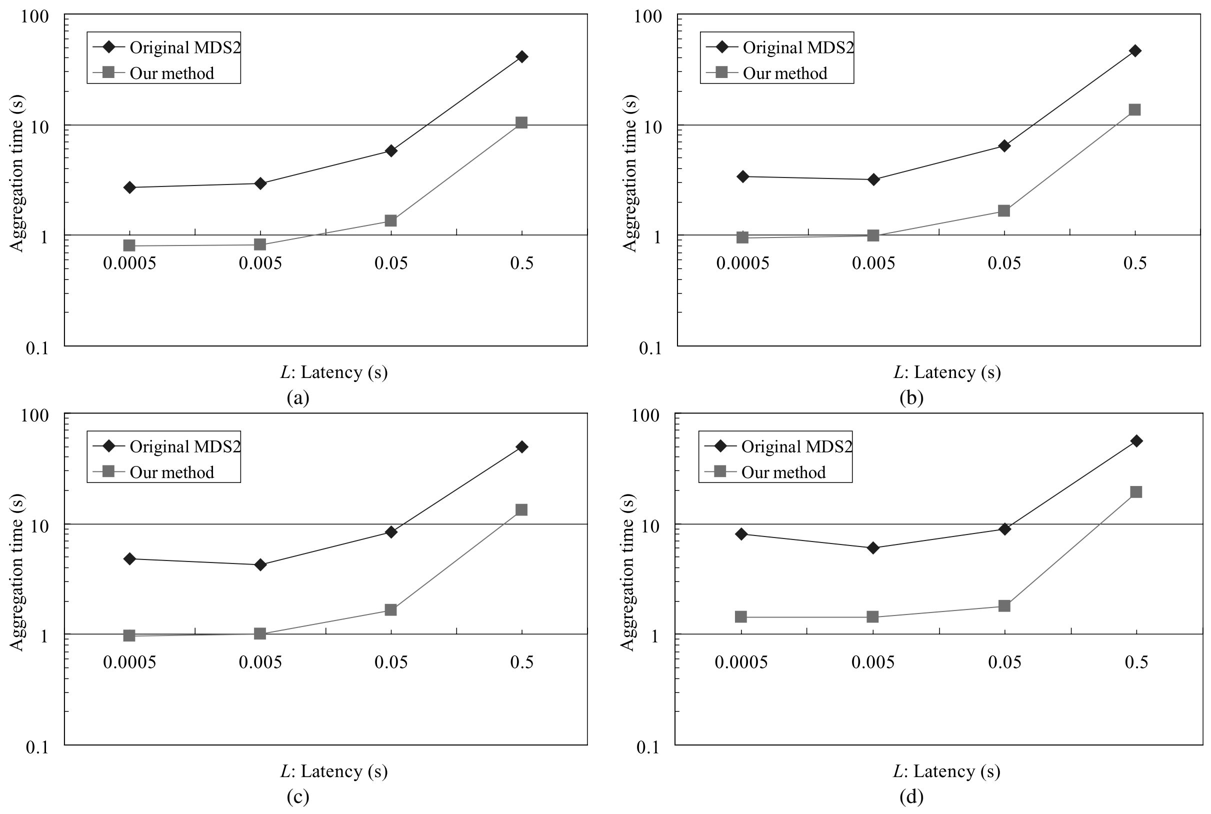


**Fig. 6** Estimated response time with 50 Mb/s bandwidth and different latency values. Results for (a) 64, (b) 128, (c) 256, and (d) 512 nodes.

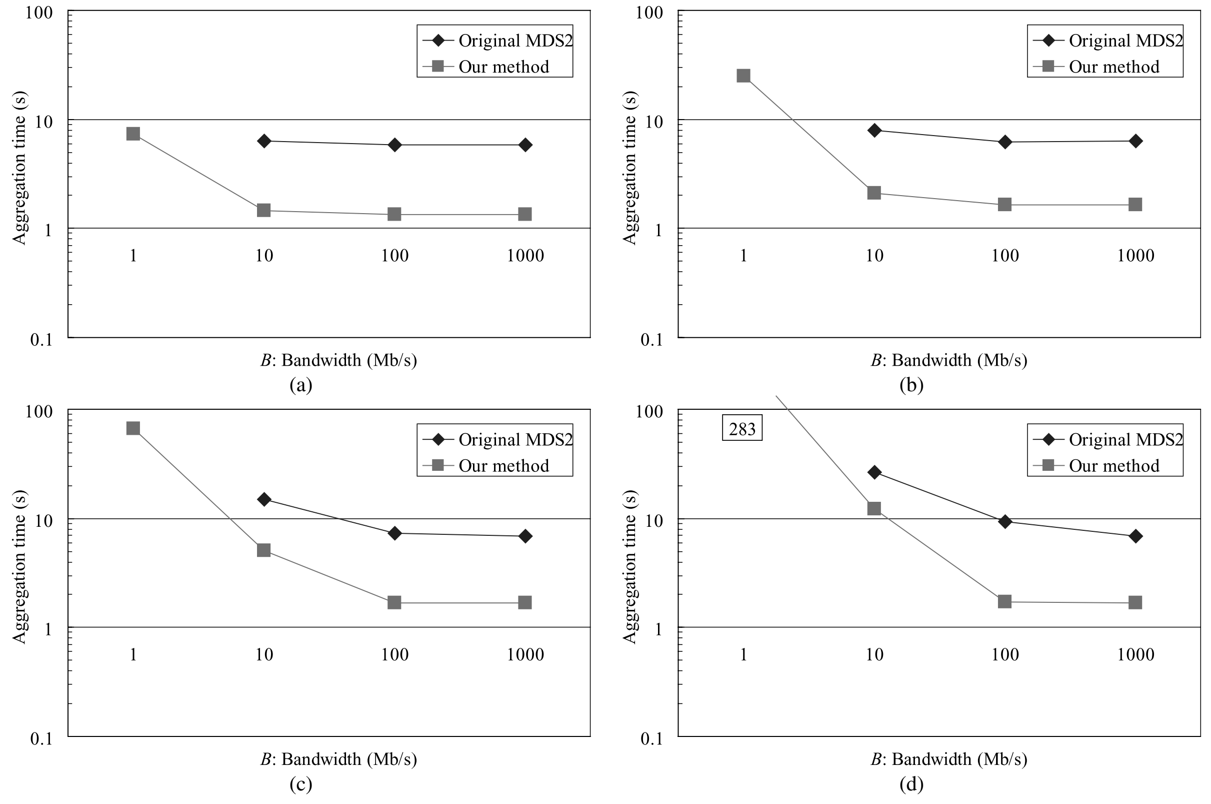


**Fig. 7** Estimated response time with 50 milliseconds latency and different bandwidth values. Results for (a) 64, (b) 128, (c) 256, and (d) 512 nodes. Some results for the original MDS2 cannot be obtained due to network congestion.

project [9], Relational Grid Monitoring Architecture (R-GMA) [25] in the European DataGrid project [8], and Hawkeye [26], part of the Condor project [5]. Although these systems have different designs, it takes approximately 30 seconds to collect resource information from 500 resources. This long response time is due to the communication overhead at the server, as we presented in Sect. 4.1.

Hawkeye provides a resource selection mechanism based on the ClassAd language and a matchmaking framework [27]. Users are allowed to identify resource properties by writing a classified advertisement (classad) expressed in the ClassAd language. According to this classad, the matchmaking framework selects a single machine on which to run a job. An extended version, which allows specifying multiple resources, is also proposed by Liu et al. [28]. A classad can include an attribute named “Rank” that represents the desirability of a match. This “Rank” attribute is similar to the evaluation value in our method, but matchmaking is done sequentially on a centralized server, degrading the scalability.

Entropia [4] demonstrate large-scale grid computing using 600 desktop PCs in a LAN environment. Thus, Entropia provides a highly scalable computing environment. However, as same as [13], their target applications require 24-hour turnaround, which can be classified into long-term applications according to our definition. Actually, a single job consists of many subjobs, each takes more than 10 minutes. Therefore, the response time is allowed to be relatively long, as compared with our method. This is also true for other monitoring systems [17], [29], [30] whose objective is to perform administrative actions such as failure recovery.

In contrast to the long-term applications mentioned above, some researchers try to execute interactive applications on the Grid environment. Talwar et al. [31] present Interactive Grids that allow users to use remote nodes for graphical interactive use. Interactive Grids have Grid Monitoring and Management Agents (GMMA) that collect the resource usage data to enforce QoS (quality of service) for applications. Although GMMA supports consolidating the monitoring data, this data reduction intends to monitor the runtime behavior of applications running on resources. For example, the raw data is aggregated into the statistics of the application at multiple points in time, allowing us to infer the behavior of applications. Wismüller et al. [32] also present a similar infrastructure for on-line monitoring and performance analysis of interactive applications. Thus, these infrastructures are intended for analyzing performance of a single application rather than the state of the Grid. Some other monitoring tools [33], [34] are also designed for the former purpose.

Agarwala et al. [35] present a low-overhead monitoring system based on kernel-level data collection. Their system decreases CPU perturbation by utilizing the /proc virtual file system of the Linux OS. It also performs data reduction by dynamic filters, which can block outgoing resource information. They show that kernel-level monitoring has small overheads and low response times on a small Linux cluster. However, their kernel-level approach is restricted to the resources controlled by the Linux OS. Therefore, we think that an application-level approach is more flexible for the Grid, which is heterogeneous in terms of OS.

In addition to the resource selection, the job submission also must be processed in a short time for rapid turnaround applications. Virtual private grid (VPG) [36] addresses this issue by eliminating authentication from the submission procedure. VPG performs authentication only when the network topology changes, and thus assumes a valid authentication for submitted jobs. A similar system is also presented by Haji et al. [18]. Although these systems achieve rapid job submission, they do not support automatic resource selection. Thus, combining our method with the VPG's authentication scheme will be a good solution for rapid turnaround grid applications.

## 6. Conclusions

We have presented a resource monitoring and selection method for rapid turnaround applications. The novelty of our method is the distributed evaluation of resources, reducing the amount of communication and the workload of the resource selection server. We implement our method using MDS2, namely a standard middleware, in order to collect more resources from different organizations.

The experimental results show that our method reduces the response time by 82% when using 64 nodes. Furthermore, the performance gain to the original MDS2 increases with the number of nodes. According to the theoretical analysis, our method will keep the response time within 1 second, up to 500 nodes in LAN environments. On the other hand, packet-level simulation results show that our method will achieve the aggregation time of less than 2 seconds up to 128 computing nodes, using a network of 10 Mb/s bandwidth and 50 milliseconds latency.

In contrast to the advantages summarized above, there are some disadvantages in our method: more workload at computing nodes and lack of the global, detailed state of a Grid. However, these issues can be resolved by using a cache mechanism and by running two MDS2 servers on computing nodes.

Future work includes support of runtime transmission of the evaluation function and development of an implementation with MDS4, which is released with Global Toolkit 4. We are also planning to integrate our method with job submission systems [18], [36] to execute rapid turnaround applications on the Grid.

**Acknowledgments**

This work was partly supported by JSPS Grant-in-Aid for Scientific Research for Young Researchers (17700060), JSPS Grant-in-Aid for Scientific Research (B)(16300006), and System Platforms Research Laboratories, NEC. We would like to thank the anonymous reviewers for their valuable comments.

## References

[1] I. Foster and C. Kesselman, eds., The Grid 2: Blueprint of a New Computing Infrastructure, second ed., Morgan Kaufmann, San Mateo, CA, 2003.

[2] I. Foster, "What is the grid? a three point checklist," 2002. http://www-fp.mcs.anl.gov/%7Efoster/Articles/WhatIsTheGrid.pdf

[3] W.T. Sullivan, D. Werthimer, S. Bowyer, J. Cobb, D. Gedye, and D. Anderson, "A new major SETI project based on project serendip data and 100,000 personal computers," Proc. 5th Int. Conf. Bioastronomy, p.729, Jan. 1997.

[4] A. Chien, B. Calder, S. Elbert, and K. Bhatia, "Entropia: Architecture and performance of an enterprise desktop grid system," J. Parallel Distrib. Comput., vol.63, no.5, pp.597–610, May 2003.

[5] M.J. Litzkow, M. Livny, and M.W. Mutka, "Condor - A hunter of idle workstations," Proc. 8th Int. Conf. Distributed Computing Systems (ICDCS'88), pp.104–111, June 1988.

[6] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke, "The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets," J. Network and Computer Applications, vol.23, no.3, pp.187–200, July 2000.

[7] W. Hoschek, J. Jaen-Martinez, A. Samar, H. Stockinger, and K. Stockinger, "Data management in an international data grid project," Proc. 1st IEEE/ACM Int. Workshop Grid Computing (GRID'00), pp.77–90, Dec. 2000.

[8] The DataGrid Project. http://www.edg.org/

[9] I. Foster and C. Kesselman, "Globus: A metacomputing infrastructure toolkit," Int. J. Supercomputer Applications and High Performance Computing, vol.11, no.2, pp.115–128, 1997.

[10] A.S. Grimshaw, W.A. Wulf, and the Legion team, "The Legion vision of a worldwide virtual computer," Commun. ACM, vol.40, no.1, pp.39–45, Jan. 1997.

[11] J. Nabrzyski, J.M. Schopf, and J. Węglarz, eds., Grid Resource Management: State of the Art and Future Trends, Kluwer Academic Publishers, Dordrecht, The Netherlands, 2003.

[12] X. Zhang, J.L. Freschl, and J.M. Schopf, "A performance study of monitoring and information services for distributed systems," Proc. 12th IEEE Int. Symp. High Performance Distributed Computing (HPDC'03), pp.270–282, June 2003.

[13] D. Kondo, A. Chien, and H. Casanova, "Resource management for rapid application turnaround on enterprise desktop grids," Proc. High Performance Networking and Computing Conf. (SC2004), Nov. 2004.

[14] Y. Kawasaki, F. Ino, Y. Mizutani, N. Fujimoto, T. Sasama, Y. Sato, N. Sugano, S. Tamura, and K. Hagihara, "High-performance computing service over the Internet for intraoperative image processing," IEEE Trans. Inf. Technol. Biomed., vol.8, no.1, pp.36–46, March 2004.

[15] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman, "Grid information services for distributed resource sharing," Proc. 10th IEEE Int. Symp. High Performance Distributed Computing (HPDC'01), pp.181–194, Aug. 2001.

[16] Y. Tanaka, H. Nakada, S. Sekiguchi, T. Suzumura, and S. Matsuoka, "Ninf-G: A reference implementation of RPC-based programming middleware for grid computing," J. Grid Computing, vol.1, no.1, pp.41–51, March 2003.

[17] W. Smith, "A system for monitoring and management of computational grids," Proc. 31st Int. Conf. Parallel Processing (ICPP'02), pp.55–62, Aug. 2002.

[18] M.H. Haji, I. Gourlay, K. Djemame, and P.M. Dew, "A SNAP-based community resource broker using a three-phase commit protocol: A performance study," Comput. J., vol.48, no.3, pp.333–346, May 2005.

[19] I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the grid: Enabling scalable virtual organizations," Int. J. High Performance Computing Applications, vol.15, no.3, pp.200–222, 2001.

[20] R. Buyya, ed., High Performance Cluster Computing, Prentice Hall PTR, Upper Saddle River, NJ, June 1999.

[21] MDS 2.2 GRIS Specification Document, "Creating new information provider," 2003. http://www.globus.org/toolkit/docs/2.4/mds/creating_new_providers.pdf

[22] W. Allcock, "GridFTP: Protocol extensions to FTP for the Grid," Global Grid Forum Recommendation GFD.20, April 2003.

[23] The Network Simulator – ns-2 –. http://www.isi.edu/nsnam/ ns/

[24] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP selective acknowledgement options," RFC (Request for comments) 2018, Oct. 1996.

[25] A. Cooke, A. Gray, W. Nutt, J. Magowan, M. Oevers, P. Taylor, R. Cordenonsi, R. Byrom, L. Cornwall, A. Djaoui, L. Field, S. Fisher, S. Hicks, J. Leake, R. Middleton, A. Wilson, X. Zhu, N. Podhorszki, B. Coghlan, S. Kenny, and D. O'Callaghan, "The relational grid monitoring architecture: Mediating information about the Grid," J. Grid Computing, vol.2, no.4, pp.323–339, Dec. 2004.

[26] Hawkeye. http://www.cs.wisc.edu/condor/hawkeye/

[27] R. Raman, M. Livny, and M. Solomon, "Matchmaking: An extensible framework for distributed resource management," Cluster Computing, vol.2, no.2, pp.129–138, 1999.

[28] C. Liu, L. Yang, I. Foster, and D. Angulo, "Design and evaluation of a resource selection framework for grid applications," Proc. 11th IEEE Int. Symp. High Performance Distributed Computing (HPDC'02), pp.63–72, July 2002.

[29] W. Allcock, J. Bester, J. Bresnahan, I. Foster, and J. Gawor, "GridMapper: A tool for visualizing the behavior of large-scale distributed systems," Proc. 11th IEEE Int. Symp. High Performance Distributed Computing (HPDC'02), pp.163–170, July 2002.

[30] M.L. Massie, B.N. Chun, and D.E. Culler, "The ganglia distributed monitoring system: Design, implementation, and experience," Parallel Comput., vol.30, no.7, pp.817–840, July 2004.

[31] V. Talwar, S. Basu, and R. Kumar, "Architecture and environment for enabling Interactive Grids," J. Grid Computing, vol.1, no.3, pp.231–250, Sept. 2003.

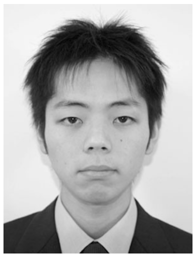
[32] R. Wismüller, M. Bubak, W. Funika, and B. Baliś, "A performance analysis tool for interactive applications on the Grid," Int. J. High Performance Computing Applications, vol.18, no.3, pp.305–316, 2004.

[33] J.S. Vetter and D.A. Reed, "Real-time performance monitoring, adaptive control, and interactive steering of computational grids," Int. J. High Performance Computing Applications, vol.14, no.4, pp.357–366, 2000.

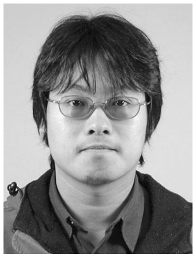
[34] D. Gunter, B. Tierney, K. Jackson, J. Lee, and M. Stoufer, "Dynamic monitoring of high-performance distributed applications," Proc. 11th IEEE Int. Symp. High Performance Distributed Computing (HPDC'02), pp.163–170, July 2002.

[35] S. Agarwala, C. Poellabauer, J. Kong, K. Schwan, and M. Wolf, "Resource-aware stream management with the customizable *dproc* distributed monitoring mechanisms," Proc. 12th IEEE Int. Symp. High Performance Distributed Computing (HPDC'03), pp.250–259, June 2003.

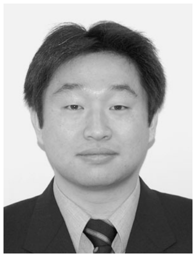
[36] K. Kaneda, K. Taura, and A. Yonezawa, "Virtual private grid: A command shell for utilizing hundreds of machines efficiently," Future Generation Computer Systems, vol.19, no.4, pp.563–573, May 2003.



**Kensuke Muraki** received the M.E. degree in information and computer sciences from Osaka University, Osaka, Japan, in 2006. He is an Engineer at the Nintendo Co., Ltd. His research interests include high-performance computing, Grid computing, and systems architecture and design.



**Yasuhiro Kawasaki** received the B.E. and M.E. degrees in information and computer sciences from Osaka University, Osaka, Japan, in 2002 and 2004, respectively. He is currently working toward the Ph.D. degree at the Department of Computer Science, Graduate School of Information Science and Technology, Osaka University. His current research interests include high-performance computing, Grid computing, and systems architecture and design. He received the Best Paper Award at the 10th International Conference on High Performance Computing (HiPC'03).



**Yasuharu Mizutani** received the M.E. and Ph.D. degrees in information and computer sciences from Osaka University, Osaka, Japan, in 2001 and 2005, respectively. He is currently a Lecturer in the Faculty of Information Science and Technology at Osaka Institute of Technology, Osaka, Japan. His research interests include parallel and distributed systems, software development tools, and performance evaluation. He received the Best Paper Award at the 10th International Conference on High Performance Computing (HiPC'03).

**Fumihiko Ino** received the B.E., M.E., and Ph.D. degrees in information and computer sciences from Osaka University, Osaka, Japan, in 1998, 2000, and 2004, respectively. He is currently an Assistant Professor in the Graduate School of Information Science and Technology at Osaka University. His research interests include parallel and distributed systems, software development tools, and performance evaluation. He received the Best Paper Award at the 2003 International Conference on High Performance Computing (HiPC'03) and the Best Paper Award at the 2004 Symposium on Advanced Computing Systems and Infrastructures (SACSIS'04).

**Kenichi Hagihara** received the B.E., M.E., and Ph.D. degrees in information and computer sciences from Osaka University, Osaka, Japan, in 1974, 1976, and 1979, respectively. From 1994 to 2002, he was a Professor in the Department of Informatics and Mathematical Science, Graduate School of Engineering Science, Osaka University. Since 2002, he has been a Professor in the Department of Computer Science, Graduate School of Information Science and Technology, Osaka University. From 1992 to 1993, he was a Visiting Researcher at the University of Maryland. His research interests include the fundamentals and practical application of parallel processing. He received the Best Paper Award at the 2003 International Conference on High Performance Computing (HiPC'03) and the Best Paper Award at the 2004 Symposium on Advanced Computing Systems and Infrastructures (SACSIS'04).