Exploiting RDMA operations for Providing Efficient Fine-Grained Resource Monitoring in Cluster-based Servers*

K. Vaidyanathan
Comp. Science and Engg.,
Ohio State University
vaidyana@cse.ohio-state.edu

H. -W. Jin †
Computer Engg. Dept.,
Konkuk University
jinh@konkuk.ac.kr

D. K. Panda
Comp. Science and Engg.,
Ohio State University
panda@cse.ohio-state.edu

Abstract

Efficiently capturing the resource usage in a shared server environment has been a critical research issue in the past several years. With the amount of resources used by each application becoming more and more divergent and unpredictable, the solution to this problem is becoming increasingly important. In the past, several researchers have come up with a number of techniques which rely on coarse-grained monitoring of resources in order to avoid the overheads associated with fine-grained monitoring. In this paper, we propose a low-overhead efficient fine-grained resource monitoring scheme using the advanced Remote Direct Memory Access (RDMA) operation provided by RDMA-enabled interconnects such as InfiniBand (IBA). We evaluate the relative benefits of our approach against traditional approaches in various environments (including micro-benchmarks as well as real applications such as an auction server based on the RUBiS benchmark and the Ganglia distributed monitoring tool). Our results indicate that our approach for fine-grained monitoring can significantly improve the overall system utilization, thereby resulting in up to 25% improvement in the number of requests the cluster-system can admit.

1 Introduction

Cluster systems consisting of commodity off-the-shelf (COTS) hardware components are becoming increasingly attractive as platforms for resource-intensive applications, primarily due to their high performance-to-cost ratio. Today, such systems are used in a variety of environments, including *compute-farms*, where users can submit jobs to be executed on the system's resources. Depending on the kind of such jobs, cluster-based systems can be broadly classified into two categories, viz., scientific supercomputers and enterprise servers. Scientific supercomputers such as the computing servers at supercomputing centers are typically used for academic as well as research oriented applications; the nodes of such systems are mainly used in a dedicated manner. Enterprise servers such as the clusters used by Google, Yahoo, Amazon, etc., are typically used for business environments; the nodes of such systems are mainly used in a shared manner, i.e., multiple applications are executed on the same set of nodes. In this paper, we concentrate on shared enterprise server environments.

Efficiently identifying the amount of resources used in such shared environments has been a critical research issue in the past several years. With the amount of resources used by each application becoming more and more divergent and unpredictable [21, 19], the solution to this problem is becoming increasingly important. This issue is especially important for financial enterprise servers. For example, several systems (e.g., Amazon) rely on the cluster resource usage information for admission control of requests, i.e., an inaccurate resource usage information could potentially lead to lost revenue for such systems.

In order to tackle this problem, in the past, several researchers have come up with a number of techniques [15, 13]. The primary idea of these techniques is to periodically monitor the resources used in the cluster and use this information to make various decisions including whether a request should be admitted, what resources should be allotted to service the request, etc. Though these approaches are generic and applicable for all environments, the main drawback with them is that they rely on coarse-grained monitoring of resources in order to avoid the overheads associated with fine-grained monitoring. Accordingly, they base their techniques on the assumption that the resource usage is consistent through the monitoring granularity (which is in the order of seconds in most cases). On the other hand, as demonstrated by recent literature, the resource usage of requests is becoming increasingly divergent making this assumption no longer valid [12].

*This research is supported in part by Department of Energy's Grant #DE-FC02-01ER25506, and National Science Foundation's grants #CNS-0403342 and #CNS-0509452; grants from Intel, Mellanox, Sun Microsystems and Linux Networx; and equipment donations from Intel, Mellanox, AMD, Apple, Sun Microsystems, PathScale, Microway and Silverstorm.

†This work was initiated when Dr. Jin was a Post-Doc and Research Scientist at OSU.

Remote Direct Memory Access (RDMA) is emerging as the central feature of modern network interconnects like InfiniBand(IBA) [7], Quadrics [4]. RDMA operations allow the network interface to transfer data between local and remote memory buffers without any interaction with the operating system or processor intervention. In this paper, we propose an approach that utilizes the advanced RDMA operations in providing efficient fine-grained resource monitoring and consequently superior resource utilization, overload control and end performance for shared server environments. We first evaluate our approach to identify its relative benefits with traditional approaches in various environments (including micro-benchmarks as well as real applications such as an auction server based on the RUBiS benchmark and the Ganglia distributed monitoring tool). Evaluations show that our approach helps in lowering the maximum and average response time of RUBiS auction benchmark by 90% and 20%, respectively. Next, we compare these approaches with traditional coarse-grained resource-monitoring techniques. Our results indicate that our approach for fine-grained monitoring can significantly improve the overall system utilization, thereby resulting in up to 25% improvement in the number of requests the cluster-system can admit.

The rest of the paper is organized as follows: Section 2 describes an overview of one-sided communication model. In Section 3, we discuss the design alternatives and implementation details. Section 4 presents the potential benefits of our approach. The experimental results are presented in Section 5. Discussion and related work are presented in Section 6. We draw our conclusions and discuss future work in Section 7.

2 Overview of One-Sided Communication Model

Many of the high-performance networks such as InfiniBand, Quadrics, etc., provide two types of communication semantics: channel semantics (send/recv communication model) and memory semantics (one-sided communication model). In channel semantics, every send request has a corresponding receive request at the remote end. Thus there is a one-to-one correspondence between every send and receive operation.

On the other hand, memory semantics follows a one-sided communication model. Here, Remote Direct Memory Access (RDMA) operations are used which allow the initiating node to directly access the memory of remote-node without the involvement of the remote-side CPU. Therefore, an RDMA operation has to specify both the memory address for the local buffer as well as that for the remote buffer. In addition, RDMA operations are allowed only on pinned memory locations thus securing the remote node from accessing any arbitrary memory location. There are two kinds of RDMA operations: RDMA Write and RDMA Read. In an RDMA write operation, the initiator directly writes data into the remote node's memory. Similarly, in an RDMA Read operation, the initiator reads data from the remote node's memory.

In this paper, we leverage the one-sided RDMA operations for achieving low-overhead fine-grained resource monitoring. Eliminating the involvement of the peer side can overcome the communication performance degradation due to CPU workload of the peer side. This also avoids any interrupt of the peer side processing.

3 Fine-Grained Resource Monitoring

The basic idea of fine-grained resource monitoring in a shared server environment is to capture the dynamic resource usage of the hosted applications. Fine-grained resource monitoring can be implemented using two approaches: back-end based monitoring and front-end based monitoring. In the former, the back-end informs the front-end node on detecting a high load. In the latter, the front-end node periodically sends a request to a resource monitoring process in the back-end to retrieve the load information. It is to be noted that when the back-end server gets a network packet from the front-end, the kernel treats it as a high priority packet and tries to schedule the resource monitoring process as early as possible. However, in the back-end resource monitoring scheme, the monitoring process sleeps for a given time interval and calculates the load information, thus decreasing its priority to be scheduled. Since the load reporting interval resolution highly depends on the operating system scheduling timer resolution, the scheduling of this back-end monitoring process is vital for sending the load responses in a timely manner. For fine-grained resource monitoring since there is a need for an immediate reply and small reporting interval resolution, front-end based resource monitoring is preferred. Previous work [18] also suggests that a front-end based approach is better than back-end based approach for fine-grained services. For these reasons, in this paper, we focus on front-end based resource monitoring.

In the following subsections, we present existing sockets-based implementations and our proposed RDMA-based design alternatives. Broadly, two ways of implementing the front-end based resource monitoring for sockets and RDMA exist: (i) Asynchronous and (ii) Synchronous. In an asynchronous approach, the load calculating phase (i.e., reading the resource usage information and calculating the current load on the back-end) and load requesting phase (i.e., requesting for load information from the front-end) are independent. On the other hand, in a synchronous approach, the back-end calculates the current load information for every request received from the front-end node.

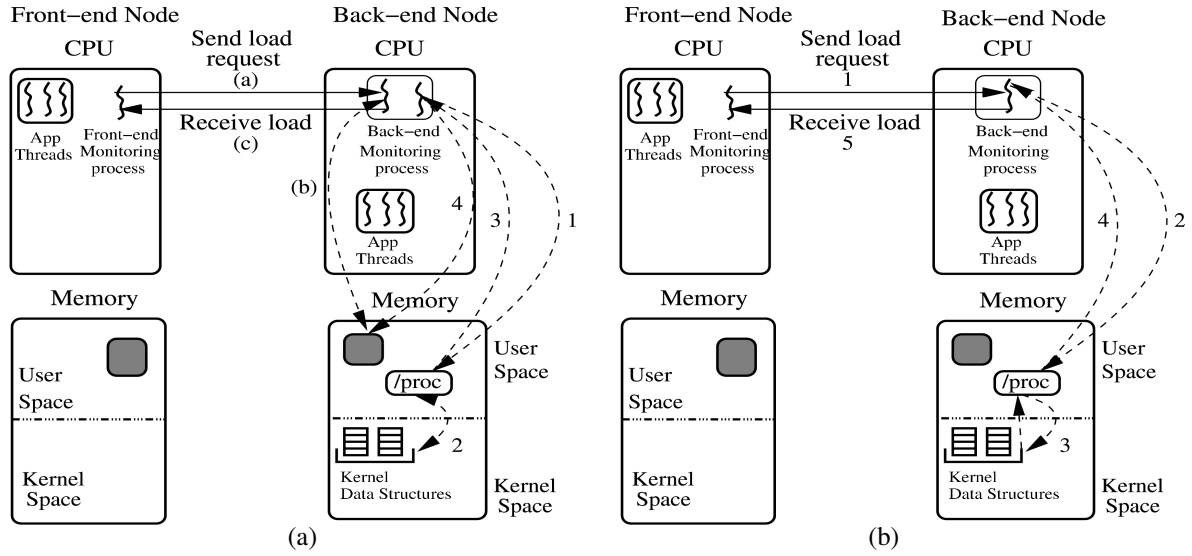


Figure 1. Sockets-based Resource Monitoring Mechanism (a) Asynchronous (b) Synchronous

3.1 Sockets-based Resource Monitoring

Traditional resource monitoring schemes use sockets communication mechanism for resource monitoring. In our implementation, we use */proc* to get the system-level statistics like information about running processes, CPU statistics, network and I/O statistics, etc.

3.1.1 Asynchronous Resource Monitoring using Sockets (Socket-Async)

In this approach, we have two processes, one running on the front-end server and the other running on the back-end server. The back-end server process consists of two threads; a load calculating thread that calculates the load information periodically and a load reporting thread that responds to load requests from the front-end servers. The sequence of steps in asynchronous resource monitoring using sockets is shown in Figure 1a. In the first step (Step 1), the load calculating thread reads */proc*. To access */proc*, a trap occurs because of file I/O in Step 2, during which the kernel calculates the system information. In Step 3, */proc* sends the monitoring information to the thread and in Step 4, the thread copies this information to a known memory location. Once this task is completed, the load calculating thread sleeps for a specific time interval T and repeats this process again. In parallel, the front-end monitoring process periodically sends a request for load information to the load reporting thread (Step a). The load reporting threads receives this request, reads the load information from the known memory location (Step b) and sends it to the front-end monitoring process (Step c).

3.1.2 Synchronous Resource Monitoring using Sockets (Socket-Sync)

This approach is very similar to the asynchronous approach, except that there is no requirement for two threads in the back-end. As shown in Figure 1b, when the front-end monitoring process sends a load request (Step 1), the back-end monitoring process calculates the load information by reading the */proc* file system (Step 2, 3 and 4) and reports this load information to the front-end monitoring process (Step 5). Thus, there is no requirement for a separate thread to calculate the load information for every time interval T.

3.2 RDMA-based Resource Monitoring

Many modern interconnects provide one-sided remote memory operations such as RDMA that allows access to remote memory without interrupting the remote CPU. In our design, we use RDMA read operation to perform efficient fine-grained monitoring.

3.2.1 Asynchronous Resource Monitoring using RDMA (RDMA-Async)

In this approach, we use two different kinds of monitoring processes running on front-end and back-end server. The back-end monitoring process handles connection management and creates registered memory regions in the user space. The front-end monitoring process periodically performs RDMA read operations on the registered memory regions to retrieve updated load information. We use the same mechanism as Socket-Async scheme for calculating the load information. The back-end monitoring process constantly calculates the relevant load information after every time T interval from */proc* and copies the load information onto the registered memory region.

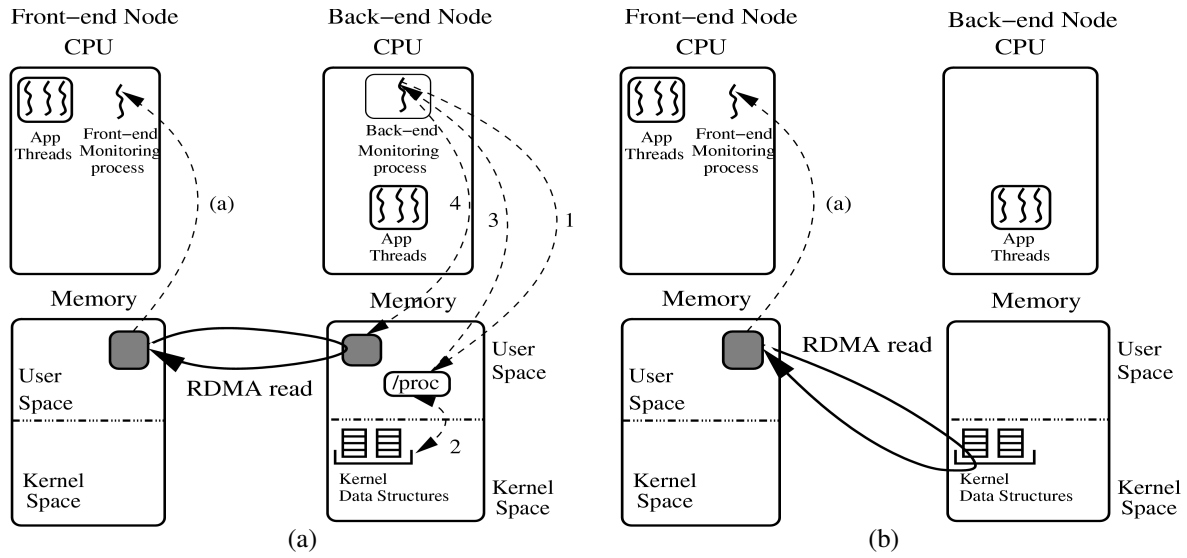


Figure 2. RDMA-based Resource Monitoring Mechanism (a) Asynchronous (b) Synchronous

The sequence of steps in RDMA-Async scheme is shown in Figure 2a. It is very similar to Socket-Async approach except that the communication is one-sided. Front-end monitoring process performs RDMA read operation (Step a) to obtain the load information.

3.2.2 Synchronous Resource Monitoring using RDMA (RDMA-Sync)

In theory, as RDMA operations are one-sided, it is not possible to have a synchronous resource monitoring approach using RDMA. However, in practice, we can achieve the accuracy of synchronous resource monitoring if the front-end node can obtain the most up-to-date load information from the kernel memory of the back-end for every request. To enable this, we register the necessary kernel data structures that hold the resource usage information, and allow the front-end monitoring process to directly retrieve this information using RDMA read as shown in Figure 2b. Such a design has two major advantages: (i) it removes the need for an extra process in the back-end server and (ii) it can exploit the detailed resource usage information in kernel space to report accurate load information. Section 4 describes these benefits in more detail.

4 Potential Benefits of RDMA-Sync

Using RDMA-Sync schemes to design and implement fine-grained resource monitoring has several potential benefits as described below.

Getting accurate load information: Due to the asynchronous nature of Socket-Async and RDMA-Async schemes, there is a delay between the time at which the back-end monitoring process updates the load information and the time at which the front-end monitoring process reads this load information. For example, if we assume that the load information is updated every T ms at the back-end server, the load information seen by the front-end monitoring process can be up to T ms old. In Socket-Sync scheme, if the server nodes are heavily loaded, the back-end monitoring process can compete for CPU with other threads in the system. This can result in huge delays in reporting the current load information to the front-end monitoring process. However, regardless of the back-end server load, RDMA-Sync scheme can report accurate load information since the front-end monitoring process directly retrieves the load information from kernel data structures without interrupting the CPU. As a result, RDMA-Sync scheme can quickly and accurately detect the load and can help to avoid overloaded conditions in several environments.

Utilizing detailed system information: While all other monitoring schemes operate at the user space, RDMA-Sync scheme operates at the kernel space. This provides several opportunities to access portions of the kernel memory which may be useful for providing system-level services. Some of them are directly exposed via */proc* interface while others like *irq_stat, dq_stat, and aven_run* are not. Though the other schemes can access these kernel data structures using a kernel module, later in the experimental section, we show some unique benefits of the RDMA-Sync scheme.

No extra thread for remote resource monitoring: All monitoring schemes except RDMA-Sync require a separate thread on the back-end server to calculate the load of the back-end node periodically. While this operation may not occupy considerable CPU, in a highly loaded server environment, it certainly competes for processor cycles. This can result in huge delays in updating the load information. Also, if the incoming traffic is extremely bursty, such de-

lays may lead to poor reconfiguration and process migration since delayed load information can give a wrong picture of current load status of the back-end servers. However, in RDMA-Sync scheme, there is no extra thread required to calculate the load information thus avoiding all the issues mentioned above.

Enhanced robustness to load: Performance of system-level services over traditional network protocols can be degraded significantly if there is a high load in the back-end. This is because both sides should get involved in communication and it is possible that the back-end monitoring process capturing the load on the back-end may never get the CPU for a long time. However, for protocols based on RDMA operations, the peer side is totally transparent to the communication procedure. Thus, the latency of both RDMA-Sync and RDMA-Async schemes is resilient and well-conditioned to load.

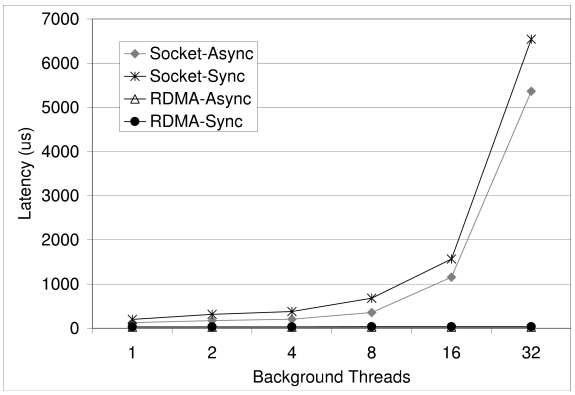


Figure 3. Latency of Socket-Async, Socket-Sync, RDMA-Async, RDMA-Sync schemes with increasing background threads

5 Experimental Results

For all our experiments we use the following two system configurations: A cluster system consisting of 8 server nodes built around SuperMicro SUPER P4DL6 motherboards and GC chipsets which include 64-bit 133 MHz PCI-X interfaces. Each node has two Intel Xeon 2.4 GHz processors with a 512 kB L2 cache and a 400 MHz front side bus and 1 GB of main memory. We used the RedHat 9.0 Linux distribution. It uses an InfiniBand network with Mellanox InfiniHost MT23108 DualPort 4x HCA adapter through an InfiniScale MT43132 twenty-four 4x Port completely non-blocking InfiniBand Switch. The IPoIB driver for the InfiniBand adapters was provided by Voltaire Incorporation [1]. The version of the driver used was 2.0.5_10.

We use 8 client nodes with two Intel Xeon 3.0 GHz processors which include 64-bit 133 MHz PCI-X interfaces, a 533 MHz front side bus and 2 GB memory. We use the RedHat 9.0 Linux distribution. Apache 2.0.48, PHP 4.3.1 and MySQL 4.0.12 was used in our experiments. Requests from the clients were generated using eight threads on each node. We use a polling time T of 50ms for resource monitoring schemes in all the experiments unless otherwise explicitly specified.

5.1 Micro-benchmarks

In this subsection, we evaluate the four schemes in terms of latency, granularity, accuracy of load information obtained and potential for extracting detailed system load information.

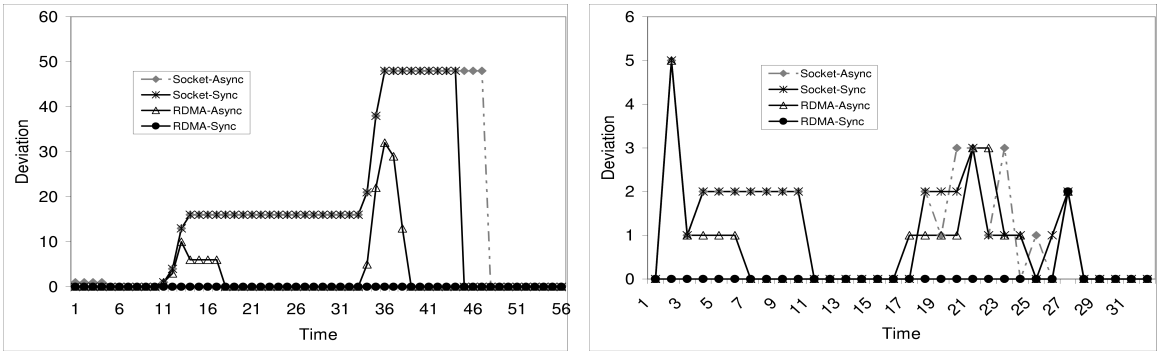
5.1.1 Latency of Resource Monitoring

In this section, we present the performance impact on the monitoring latency of the four schemes with loaded conditions in the cluster-based servers. We emulate the loaded conditions by performing background computation and communication operations on the server while the front-end monitoring process monitors the back-end server load. This environment emulates a typical shared server environment where multiple server nodes communicate periodically and exchange messages, while the front-end node, which is not as heavily loaded, attempts to get the load information from the monitoring process on the heavily loaded servers.

The performance comparison of Socket-Async, Socket-Sync, RDMA-Async and RDMA-Sync for this experiment is shown in Figure 3. We observe that the monitoring latency of both Socket-Async and Socket-Sync increase linearly with the increase in the background load. On the other hand, the monitoring latency of RDMA-Async and RDMA-Sync scheme which use one-side communication, stays the same without getting affected by the background load. These results show the capability of one-sided communication primitives in a cluster-based server environment.

5.1.2 Granularity of Resource Monitoring

We present the impact on the performance of running applications with respect to increasing granularity of resource monitoring of all four schemes. In this experiment, application performs basic floating-point operations and reports the time taken. We report the average application delay normalized to the application execution time for each of the schemes as we vary the granularity from 1 ms to 1024 ms as shown in Figure 4. We observe that the application performance degrades significantly when Socket-Async, Socket-Sync and RDMA-Async schemes are running in the background at smaller granularity such as 1 ms and 4 ms. Since the Socket-Async scheme uses two threads for resource monitoring in the back-end server, we find that this scheme affects the application performance significantly in comparison with other schemes. In RDMA-Async scheme, due to the presence of the back-end monitor-



(a) (b)

Figure 5. Accuracy of Load information (a) Number of threads running on the server (b) Load on the CPU

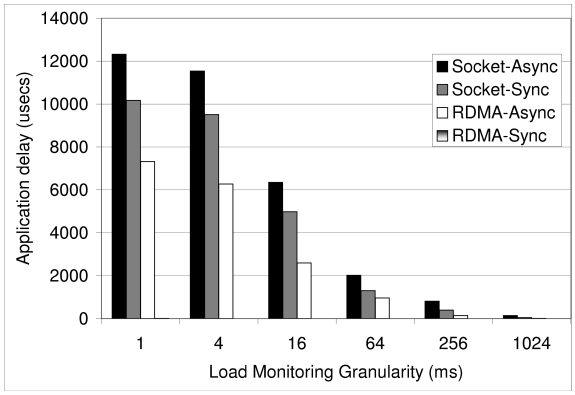


Figure 4. Impact on application performance with Socket-Async, Socket-Sync, RDMA-Async, RDMA-Sync schemes

ing process, we see that the application performance degradation is lesser in comparison with two-sided Socket-Sync scheme. However, we find that there is no performance degradation with RDMA-Sync scheme due to the fact that there are no processes running on the back-end server to affect the application performance.

5.1.3 Accuracy of Load Information

In this experiment, we analyze the accuracy of the load information obtained from the four schemes. In order to be uniform across all four schemes, we design the experiment in the following way. We run all four schemes simultaneously and monitor the load information. In addition, a kernel module on the back-end server periodically reports the actual load information at a finer granularity. To emulate loaded conditions, we fired client requests to be processed at the back-end server. We compare these numbers against the load information reported by the kernel module and plot the deviation between these two values.

Figure 5a shows the deviation of the number of threads running on the server with respect to the numbers reported by all four schemes. We see that all four schemes report the same values initially since there was no load on the server. However, as the load on the server increases, we see that Socket-Async, Socket-Sync and RDMA-Async show deviations with respect to the number of threads reported by the kernel module. On the other hand, RDMA-Sync scheme consistently reports no deviation. Further, we observe that both Socket-Async and Socket-Sync schemes show large deviations when the back-end server is heavily loaded. Since sockets is a two-sided communication protocol, as the load on the back-end server increases, the latency for capturing the number of threads running also increases leading to such inaccuracies.

Figure 5b shows the accuracy of the CPU load information reported by all four schemes in comparison with the actual CPU load. We perform the experiment in a similar way as explained above. We find that Socket-Async, Socket-Sync and RDMA-Async schemes show deviations in comparison with the actual load while RDMA-Sync scheme reports very few deviations. Since CPU load fluctuates more rapidly in comparison with number of threads in the system, we see that RDMA-Async scheme also reports inaccurate load information. Socket-Async and Socket-Sync schemes, due to the reasons mentioned above, report stale CPU load values leading to large deviations.

5.1.4 Detailed System Information

In this experiment, we evaluate our four schemes in terms of their ability to obtain detailed system information with finer granularity. To explore this feature, we measure the number of interrupts pending on CPUs of the servers. For our evaluation, we use the *irq_stat* kernel data structure, which maintains the number of software, hardware and bottom-half pending interrupts on each of the CPUs. We use all

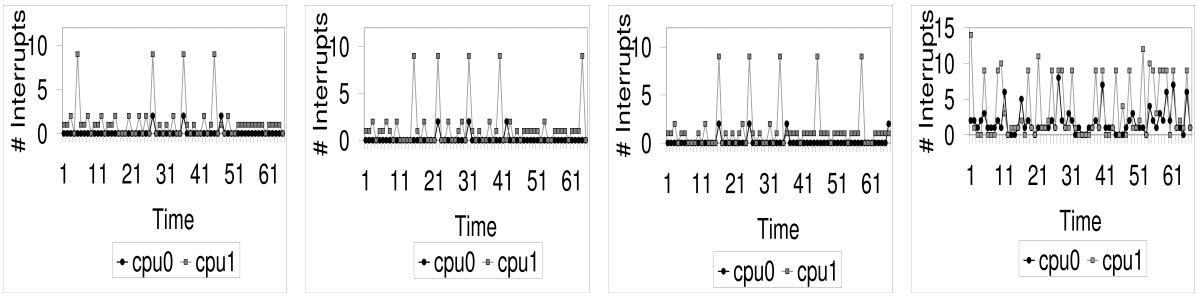


Figure 6. Number of Interrupts reported on both CPUs: (a) Socket-Async (b) Socket-Sync (c) RDMA-Async (d) RDMA-Sync

four schemes to report these values to the front-end monitoring process. Since the data structure is available at the kernel space, we use a kernel module to expose this to user-space, so that Socket-Async, Socket-Async and RDMA-Async schemes can report this information.

We see that the three schemes, Socket-Async, Socket-Sync and RDMA-Async as shown in Figure 6a, 6b and 6c report less and infrequent interrupts in comparison with the RDMA-Sync scheme as shown in Figure 6d. As mentioned above, an user process triggers the kernel module to report the interrupt information for these three schemes. However, if there are pending interrupts on the CPUs and an user process, operating system would give a higher priority to schedule the interrupts rather than the user process. Furthermore, in a uni-processor kernel, the operating system may complete all the interrupt handling and then pass the control to the user process. However, since there is no such requirement for RDMA-Sync scheme, we observe that this scheme reports interrupt information more accurately. Interestingly, RDMA-Sync scheme reports more interrupts (in terms of the number of interrupts) in comparison with the other three schemes. Moreover, the number of interrupts reported on the second CPU by RDMA-Sync scheme is consistently higher in comparison with the numbers reported by all other schemes.

5.2 Application-Level Evaluation

In this section, we study the benefits of our resource monitoring schemes with two well known applications. (i) Web Servers and (ii) Ganglia - a cluster management tool. Next, we compare the performance of the proposed fine-grained resource monitoring schemes against the traditional approaches.

5.2.1 Evaluation with Web Servers

Request patterns seen over a period of time inside the hosting centers that provide web services, may vary significantly in terms of the popularity of content. In addition, requests themselves may have varying computation time. Small documents get served quickly while large documents take more time to get transferred. Similarly dynamic webpages take varying computation time depending on the type of the client request. Due to these complex issues, balancing the requests to efficiently utilize all the nodes in hosting centers is a challenging task. We chose a popular algorithm used by IBM WebSphere [3], to evaluate the impact of our resource monitoring schemes in load balancing. IBM WebSphere utilizes load information such as CPU, memory, network and connection load, assigns appropriate weights to these load indices and calculates the average load of the server. The least loaded servers are chosen for servicing the request.

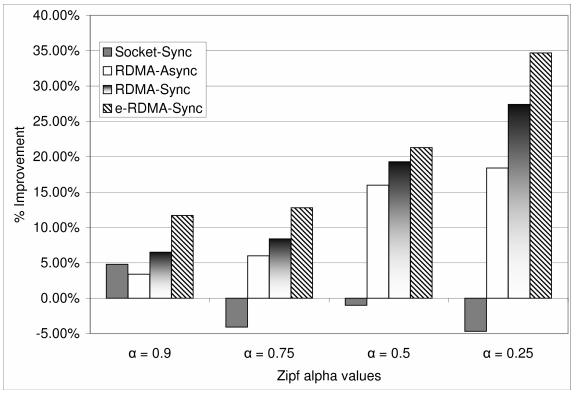


Figure 7. Throughput Improvement of Socket-Async, Socket-Sync, RDMA-Async, RDMA-Sync, e-RDMA-Sync schemes with RUBiS and Zipf Trace

Cluster-based Server with RUBiS: We evaluate our schemes in a cluster-based server environment using a RUBiS auction benchmark [5, 6] developed by Rice University. The benchmark simulates a workload similar to a typ-

Table 1. RUBiS Benchmark

Query	Average Response Time					Maximum Response Time				
	Socket Async	Socket Sync	RDMA Async	RDMA Sync	e-RDMA Sync	Socket Async	Socket Sync	RDMA Async	RDMA Sync	e-RDMA Sync
Home	3	3	3	3	2	416	274	36	33	31
Browse	3	3	3	3	2	495	348	197	81	45
BrowseRegions	6	6	6	5	5	392	206	249	41	32
BrowseCatgryReg	17	17	18	16	14	265	278	210	74	66
SearchItemsReg	4	4	4	4	3	150	180	78	43	32
PutBidAuth	3	3	3	3	2	99	231	38	30	20
Sell	4	4	3	2	2	373	264	19	21	21
About Me (auth)	3	3	3	3	2	178	220	32	35	32

ical e-commerce website. It implements the typical functionality of auction sites such as selling, browsing and bidding. We modified the client emulator to fire requests to multiple servers and we use the algorithm mentioned above to evaluate the resource monitoring schemes. In order to understand the benefits of detailed system information, we added an e-RDMA-Sync scheme that utilizes system load and also pending interrupts on the CPUs for choosing the least-loaded servers. All other schemes use only system load for choosing the least-loaded servers. Table 1 reports average and maximum response time of several queries of the RUBiS benchmark. We find that, both RDMA-Sync and e-RDMA-Sync schemes perform consistently better than the other three schemes. Since the maximum response time is considerably low for RDMA-Sync scheme in comparison with Socket-Async, Socket-Sync and RDMA-Async schemes, it validates the fact that completely removing the need for another process on the server brings down the maximum response time. The improvement we realize for queries like BrowseRegions, Browse is close to 90% for RDMA-Sync and e-RDMA-Sync scheme. Other queries like BrowseCategoriesInRegions and SearchCategoriesInRegion show benefits up to 80% for both RDMA-Sync and e-RDMA-Sync scheme. In addition, we also observe that there is considerable improvement in the average response time of the queries for RDMA-Sync and e-RDMA-Sync schemes in comparison with other schemes. Also, we see that e-RDMA-Sync scheme consistently performs better than RDMA-Sync scheme showing the benefits of using detailed system information for performing effective and fine-grained services.

Cluster-based Server with RUBiS and Zipf Trace: In order to show the maximum potential of fine-grained resource monitoring, we design an experiment where cluster-based servers host two web services. We use a Zipf trace with varying α value. According to Zipf law, the relative probability of a request for the ith most popular document is proportional to $1/i^{\alpha}$, where α determines the randomness of file accesses. Higher the α value, higher is the temporal locality of the document accessed.

The experiment is designed in the following manner. We run the RUBiS benchmark and Zipf traces simultaneously and use all five schemes namely Socket-Async, Socket-Sync, RDMA-Async, RDMA-Sync and e-RDMA-Sync for resource monitoring. We fix the RUBiS benchmark and vary the α value for the Zipf trace from 0.25 to 0.9. As mentioned earlier, higher α values mean the workload has a high temporal locality. We report the total throughput improvement in comparison with the Socket-Async scheme for each of these traces separately as shown in Figure 7. We can observe that in the case of Zipf trace with α value 0.25, both RDMA-Sync and e-RDMA-Sync schemes achieve a performance improvement of up to 28% and 35% respectively. For smaller α values, we see a considerable performance improvement. This is due to the fact that there are lots of requests with different resource requirements and these requests are forwarded to appropriate servers in a timely manner. As α value increases, the number of requests with different resource requirements decreases resulting in an increase in the temporal locality of the documents. Hence the load on all the servers are already well distributed leading to lesser performance gains.

5.2.2 Evaluation with Ganglia

The second application we use to evaluate is Ganglia [2]. Ganglia is a scalable distributed monitoring system for clusters and monitors several system-level statistics such as CPU, memory and network usage, etc. In addition, Ganglia uses a metric tool known as gmetric, which allows users to specify any arbitrary metric to be monitored apart from the default metrics. For our evaluation, we use this metric to support fine-grained monitoring for Ganglia. Our resource monitoring schemes capture detailed system information and reports to gmetric which in turn informs all ganglia servers.

For our experiments, as we concluded in Section 5.2.1, RUBiS benchmark with e-RDMA-Sync perform the best in comparison with all other schemes, we ran the RUBiS benchmark utilizing this e-RDMA-Sync scheme along with Ganglia monitoring system. The motivation behind this experiment is to see if any of our resource monitoring schemes affect the performance of the RUBiS benchmark.

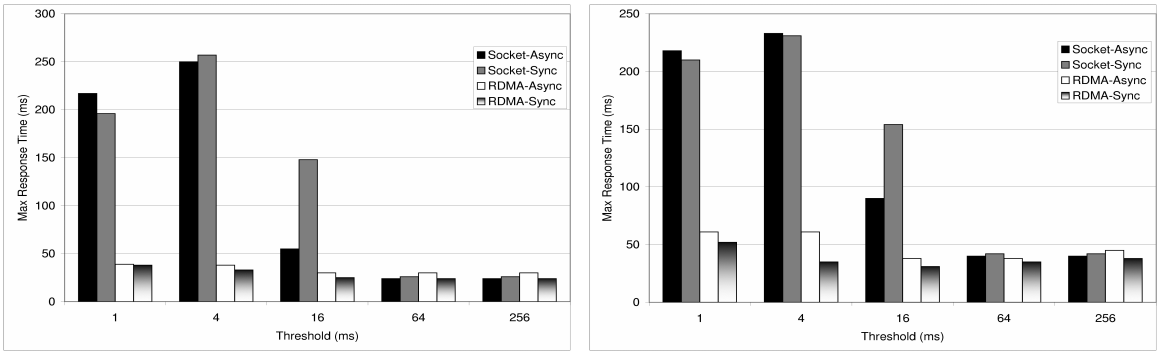


Figure 8. Response Time of RUBiS benchmark with Ganglia (a) SearchItemInCategories (b) Browse

In the background, we use ganglia to monitor the cluster and gmetric which uses one of four schemes (Socket-Async, Socket-Sync, RDMA-Async and RDMA-Sync) to perform fine-grained monitoring. Figure 8 shows the maximum response time of two queries of a RUBiS benchmark. Figure 8a reports the maximum response time of SearchItemInCategories query and Figure 8b reports the maximum response time of Browse query. As we see in Figure 8a, when gmetric uses Socket-Async or Socket-Sync to perform fine-grained resource monitoring, the maximum response time for the query is close to $250ms$ when the resource monitoring threshold is 1 or 4 ms. However, we find that the maximum response time is unaffected when gmetric uses RDMA-Async or RDMA-Sync scheme to perform fine-grained resource monitoring. We see similar trends in Figure 8b. This validates that RDMA-based resource monitoring does not affect the performance of other applications as compared to a two sided communication protocol like sockets.

5.2.3 Fine-grained Vs Coarse-grained Monitoring

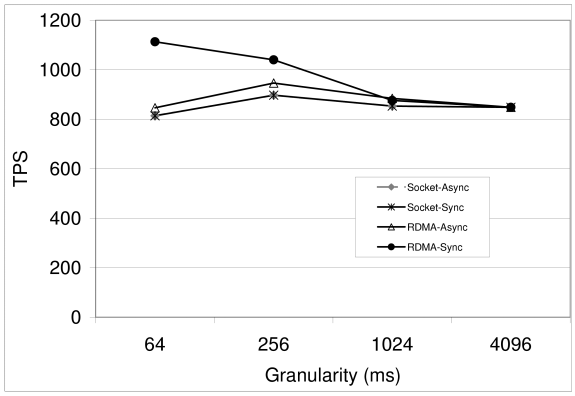


Figure 9. Fine-grained vs Coarse-grained Monitoring

In order to understand the impact of granularity of resource monitoring with applications, we evaluate the performance for different load fetching granularity from 64 $m$secs to 4096 $m$secs. Figure 9 shows the throughput performance of a RUBiS benchmark and Zipf trace with α value 0.5 running simultaneously for different load fetching granularity. We see that the throughput increases for decreasing granularity for RDMA-Sync scheme. With granularity 1024 m secs, all four schemes report comparable performance. As the granularity decreases to 64 $m$secs, we see a performance degradation for Socket-Sync and Socket-Async schemes. Thus, traditional resource monitoring approaches based on sockets cannot be used for fine-grained resource monitoring. On the other hand, we see an improvement close to 25% for RDMA-Sync scheme compared to the rest of the schemes when the granularity is 64 $m$secs showing the performance benefits of fine-grained resource monitoring with applications. Thus, our results indicate that fine-grained monitoring can significantly improve the overall utilization of the system and accordingly lead to up to 25% improvement in the number of requests the cluster-system can admit.

6 Discussion and Related Work

Several researchers have proposed the feasibility and potential of cluster-based servers [13, 17] for scalability and availability of resource-intensive distributed applications. In the past, researchers have proposed coarse-grained monitoring approaches [15, 13] in order to avoid the overheads associated with fine-grained monitoring for such environments. In this paper, we propose an approach for providing a low-overhead fine-grained resource monitoring services which can complement the cluster-based server infrastructure and result in better performance of system-level services.

Researchers have proposed and evaluated various load balancing policies using load information for cluster-based network services [11, 18]. Our proposed scheme is applicable to all the schemes that uses monitored information. In

addition, some of the existing load-balancing schemes can be enhanced further using the detailed system information provided by our scheme. Several others have focused on the design of adaptive systems that can react to changing workloads in the context of web servers [10, 14, 16]. Our schemes are also applicable in these environments which uses monitored load information for reconfiguration of resources.

Many of the modern interconnects such as IBA also provide features such as hardware multicast which can make our solutions highly scalable. For example, the back-end server node can use *hardware multicast* to inform a group of front-end dispatchers about its status. These solutions are known to scale well in large scale clusters. However, these features use channel semantics as described in Section 2. Hence such solutions are not completely one-sided removing some of the benefits of our design. Due to the fact that the kernel data structures are shared among different nodes in the cluster environment, our solution demands the need for accessing these memory regions remotely. Exposing such memory regions can lead to security issues (e.g. a remote node can update some internal kernel data structures). However, we mark these memory regions as read-only thus avoiding the risk of modifying these memory regions remotely.

7 Conclusions and Future Work

In this paper, we propose an approach for achieving efficient fine-grained resource monitoring using the advanced RDMA operation of current generation intelligent network adapters such as IBA, etc. We evaluate the benefits of our proposed approach relative to traditional approaches in various environments (including micro-benchmarks as well as real applications such as an auction server based on the RUBiS benchmark and the Ganglia distributed monitoring tool). Our results indicate that our approach for fine-grained monitoring can significantly improve the overall utilization of the system and accordingly lead to up to 25% improvement in the number of requests the cluster-system can admit.

Dynamic reconfiguration of resources has been studied in the context of nodes [9, 8], file systems [20] and storage environments. Accurate monitoring of resources is critical for efficient resource utilization in these environments. We plan to extend the knowledge gained in this study to implement a full-fledged reconfiguration module coupled with accurate resource monitoring.

References

[1] Breaking through the Bottleneck. `http://www.voltaire.com`.

[2] Ganglia Cluster Management System. http://ganglia.sourceforge.net/.

[3] Load balancing for e-network communication servers. www.redbooks.ibm.com/redbooks/pdfs/sg245305.pdf.

[4] Quadrics Supercomputers World Ltd. http://www.quadrics.com/.

[5] Rubis: Rice university bidding system. http://rubis.objectweb.org.

[6] C. Amza, E. Cecchet, A. Chanda, A. Cox, S. Elnikey, R. Gil, J. Marguerite, K. Rajamani, and W. Zwaenepoel. Bottleneck characterization of dynamic web site benchmarks. In *Technical Report TR02-391,Rice University*, 2002.

[7] Infiniband Trade Association. http://www.infinibandta.org.

[8] P. Balaji, S. Narravula, K. Vaidyanathan, H. W. Jin, and Dhabaleswar K. Panda. On the Provision of Prioritization and Soft QoS in Dynamically Reconfigurable Shared Data-Centers over InfiniBand. In *the Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2005.

[9] P. Balaji, K. Vaidyanathan, S. Narravula, K. Savitha, H. W. Jin, and D. K. Panda. Exploiting Remote Memory Operations to Design Efficient Reconfiguration for Shared Data-Centers. In *Workshop on Remote Direct Memory Access (RDMA): Applications, Implementations, and Technologies (RAIT)*, San Diego, CA, Sep 20 2004.

[10] J. Carlstrom and R. Rom. Application-aware admission control and scheduling in web servers. In *Proceedings of the IEEE Infocom 2002*, June 2002.

[11] E. Carrera and R. Bianchini. Efficiency vs. portability in cluster-based network servers. In *Proceedings of the 8th Symposium on Principles and Practice of Parallel Programming, Snowbird, UT*, 2001.

[12] Jeffrey S. Chase, Darrell C. Anderson, Prachi N. Thakar, Amin Vahdat, and Ronald P. Doyle. Managing energy and server resources in hosting centres. In *Symposium on Operating Systems Principles*, 2001.

[13] Armando Fox, Steven D. Gribble, Yatin Chawathe, Eric A. Brewer, and Paul Gauthier. Cluster-based scalable network services. In *Symposium on Operating Systems Principles*, 1997.

[14] S. Lee, J. Lui, and D. Yau. Admission control and dynamic adaptation for a proportionaldelay diffserv-enabled web server. In *Proceedings of SIGMETRICS*, 2002.

[15] O. Othman, J. Balasubramanian, and D.C. Schmidt. The Design of an Adaptive Middleware Load Balancing and Monitoring Service . In *Proceedings of the Third International Workshop on Self-Adaptive Software (IWSAS)*, June 2003.

[16] G. Porter and R.H. Kaltz. Effective Web Service Loadbalancing through Statistical Monitoring. In *SelfMan 2005, IFIP/IEEE International Workshop on Self-Managed Systems and Services*, May 2005.

[17] Yasushi Saito, Brian N. Bershad, and Henry M. Levy. Manageability, availability and performance in porcupine: A highly scalable, cluster-based mail service. In *Symposium on Operating Systems Principles*, 1999.

[18] K. Shen, T. Yang, and L. Chu. Cluster load balancing for fine-grain network services. In *Proc. of International Parallel and Distributed Processing Symposium. FL. April*, 2002.

[19] W. Shi, E. Collins, and V. Karamcheti. Modeling Object Characteristics of Dynamic Web Content. *Special Issue on scalable Internet services and architecture of Journal of Parallel and Distributed Computing (JPDC)*, Sept. 2003.

[20] K. Vaidyanathan, P. Balaji, H. W. Jin, and D. K. Panda. Workload-driven Analysis of File Systems in Shared Multi-Tier Data-Centers over InfiniBand. In *8th Workshop on Computer Architecture Evaluation using Commerical Workloads (CAECW-8); In conjunction with International Symposium on High Performance Computer Architecture (HPCA)*, 2005.

[21] J. Yin, L. Alvisi, M. Dahlin, and A. Iyengar. Engineering Web Cache Consistency. *ACM Transactions on Internet Technology, 2:3,*, August. 2002.