J. Parallel Distrib. Comput. 70 (2010) 732–742

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

A session key caching and prefetching scheme for secure communication in cluster systems☆

Manhee Lee *, Baik Song An, Eun Jung Kim **

Department of Computer Science and Engineering, Texas A&M University, College Station, TX 77843, United States

ARTICLE INFO

Article history:
Received 5 January 2009
Received in revised form
19 October 2009
Accepted 23 November 2009
Available online 24 December 2009

Keywords:
Security
Cluster
Scheduling
Cluster interconnects
Encryption
Authentication

ABSTRACT

With the widespread use of cluster systems and ever increasing threat to computer security, it becomes more necessary to design and build secure cluster systems. Most cluster systems rely on security products like firewalls for their security, but they cannot guarantee security of intra-cluster communications, which can be a weak spot that hackers exploit for further security attacks. A recent study by Lee and Kim (2007) [22] proposed a security framework to protect intra-cluster communications by encrypting and authenticating all packets with fine-grained security where any two communicating processes dynamically generate and share a cryptographic key, called a session key. However, the fine-grained security scheme can incur serious performance degradation in large-scale cluster systems since it may take a long time to access session keys. To solve this problem, we propose to incorporate a session key cache inside a cluster interconnect card to speed up accesses to the session keys and build an analytical cluster traffic model to estimate the behavior of the cache in large-scale cluster systems. For further performance improvement, we propose a prefetching scheme speculating job scheduler's decision without OS interventions. Simulation results indicate that the session key cache with the prefetching scheme decreases the network latency by 50% on average, compared to the configurations without the enhancements.

© 2009 Elsevier Inc. All rights reserved.

1. Introduction

Cluster systems have emerged as the most cost-effective solution for various applications and services. As many institutions like banks, military, and government agents that can be targets of intensified terrors adopt cluster systems, the importance of cluster security increases significantly. Besides, ordinary hackers are also attracted to cluster systems because they can stage a massive security attack by utilizing abundant resources such as huge computation power, large disk space, and high speed networks. Unfortunately, traditional security countermeasures like firewalls and intrusion detection systems (IDS) are not sufficient to provide complete security for cluster systems, as witnessed in the recent illegal break-ins to the most powerful cluster systems, some of which belong to the San Diego Supercomputer Center and the National Center for Supercomputing Applications [13,21].

Among many possible attacks, we focus on both physical and software attacks on cluster communications.[1] Physical attacks on cluster communications capture or modify data from cluster interconnects through a snooping device attachable to the interconnects. Since all cluster communication messages are plaintext, one successful physical attack will leak a significant amount of information from cluster systems. In a recent survey questioned to supercomputer administrators [26], 8% of respondents reported that there were unlawful physical approaches to their systems, and another 8% told that someone had tried to bribe inside personnel to help them infiltrate cluster systems. Considering tremendous aftereffects of physical attacks, the percentages do not seem negligible at all. Moreover, as shown in [15], cluster interconnects can be used to connect multiple clusters and storage systems, inevitably having their cables vulnerable to possible physical attacks. Therefore, we believe that the possibility of physical attacks cannot be overlooked any more, so the attacks should be prevented with great care.

Meanwhile, software attacks on cluster communication are done by a hacker who acquires a cluster account illegitimately and

☆ This work was supported in part by NSF grants CCF-0541360, CCF-0541384 and NSF CAREER award (CCF-0845998).
* Corresponding author.
** Principal corresponding author.
E-mail addresses: manhee@cs.tamu.edu (M. Lee), baiksong@cs.tamu.edu (B.S. An), ejkim@cs.tamu.edu (E.J. Kim).

[1] Cluster communication in this paper is equivalent to intra-cluster communication.

0743-7315/$ – see front matter © 2009 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2009.11.007

logs into a cluster node as a normal cluster user. If the hacker injects or sniffs packets without constraints, the cluster system will be compromised seriously. Note that preventing software attacks on cluster communication can be very effective in bolstering cluster security because the hacker may prefer such attacks to other software attacks. Moreover, the attacks on cluster communications may be the last option for the hacker; thanks to enhanced cluster security by using partitioning, sandboxing, multi-level security, or other confinement schemes, other possible software attacks beyond predefined resources can be prevented. Consequently, we believe that cluster communications need to be protected effectively against software attacks as well as physical attacks.

To prevent these attacks, [22] proposed a comprehensive security framework for InfiniBand cluster systems. The framework uses a fine-grained security scheme where any two communicating processes or queue-pairs in the InfiniBand architecture (IBA) share a session key dynamically to encrypt/authenticate all the communications. Since in their framework the security function resides in cluster interconnect cards (CICs), not in host CPUs, it causes no additional performance overhead to host CPUs. However, they assumed that session keys stored in the CIC are accessible for cryptographic operations without any delay. This assumption of zero delay in session key access time is unrealistic for the following reasons. If the session keys are stored in an off-chip memory in the CIC, every packet arrival incurs an additional memory access, resulting in a very long packet processing time. Even if a data cache in an embedded processor in the CIC can be used to store the session keys, a cryptographic hardware that is usually implemented as a separate hardware module cannot share the data cache with the embedded processor easily. Furthermore, since the temporal and spatial locality for data and session keys will be different, sharing one cache may hurt the overall performance by evicting either of them prematurely before exploiting its locality. As a remedy for this, the hardware cryptographic unit can have its own cache to store recently used session keys, referred to as *session key cache (SKC)*. In order for the SKC to be used in the CIC, a comprehensive study is necessary to answer the following important questions:

- How will the CIC look? Where are the hardware cryptographic unit and SKC located? What are their hardware costs in terms of area and power consumption? If physical attacks are possible, the CIC can be vulnerable to the attacks, too. Then, what kind of security technology should be applied to the CIC to protect the CIC from the physical attacks?
- Is a small SKC scalable to large-scale cluster systems? Will there exist any analytic model to estimate the proper size of the SKC? What is the relationship between the size and the hit rate of the SKC?
- If there are multiple applications (or processes) running on a node, does this affect hit rates of the SKC? If so, is there a solution to maintain a high SKC hit rate? Does the solution need to closely work with the OS's schedulers? Will the solution work well consistently with various cluster schedulers?

We attempt to answer these questions by doing rigorous research on cluster traffic simulation and analysis. We first provide a detailed design of the CIC. Second, to estimate the behavior of the SKC in large-scale clusters, we develop an analytical model based on an observation that cluster traffic patterns are well fitted to exponential distributions [41]. We use this model to find out the relationship between SKC sizes and hit rates of the SKC. Third, to maintain a short session key access time even in a multitasking environment we propose to incorporate a prefetch buffer in the CIC to fetch session keys ahead of time by predicting the next scheduling decision. Finally, to evaluate the new CIC design, we developed a trace-driven simulator by modifying a cycle-accurate cluster network simulator that was used in [19]. We captured real traces in two ways: from a real machine, an SGI Altix 3700 supercomputer and a Simics full-system simulator with GEMS, using NAS parallel and SPECjbb2000 benchmarks. We also developed five schedulers: Linux local, spin-block (SB) [1], dynamic coscheduling with spin-block (DCS-SB) [32,39], periodic boosting (PB) [32], and gang scheduler [10,14].

Simulation results show that an SKC reduces the security overhead on network latency by 50% on average, compared to non-SKC configurations. In multitasking environments, the session key prefetching scheme is helpful in reducing network latency by 5% or more on average. An analytic cache simulation estimates that a 16-Kbyte SKC can support up to thousands of session keys with high SKC hit rates.

The remainder of the paper is organized as follows: Section 2 explains a threat model of our research, and describes major challenges in providing high performance cluster security. In Section 3, we propose the design of an SKC and a prefetch buffer, an analytical model for NAS parallel benchmarks, and a prefetching scheme. Simulation results and their analysis are presented in Section 4, followed by related work and concluding remarks in Sections 5 and 6.

2. High performance cluster security

In this section, we describe the threat model and the architectural environment of our research, and then we discuss the CIC approach for high-performance cluster security by comparing three possible scenarios: host CPU, security coprocessor, and the CIC.

2.1. Security assumptions

We would like to begin with defining a threat model in order to clarify what kinds of threats we want to defend against. As for physical attacks, we assume the following constraints. We assume that an intruder can access a cluster system physically and he wants to be as unobtrusive as possible in order to go unnoticed. So he prefers installing a snooping device in the system to accessing a console connected to the cluster system to get a root privilege or copy data from the console so that he can steal data from the system secretly for a long time. Many previous studies have investigated physical attack prevention mechanisms for a single system or each cluster node [11,24,35,42]. So we assume it is impossible to capture or modify data from processor, memory system, memory bus, or hard drives. In addition, based on the studies of security coprocessor and smartcard, we also assume that the CIC has a tamper-resistant mechanism so that some part of the CIC can be physically attack-proof. As shown in Fig. 1, we assume that the tamper-resistant mechanism is applied to the CIC controller containing an embedded processor and caches, while other components like the CIC memory and its memory bus are insecure, which is a common assumption in hardware security research [36,38,12,9]. Other cluster components including switch/router and cables are also assumed to be vulnerable to physical attacks.

As for software attacks, we assume that a hacker compromised a cluster node, so he has a full access to cluster interconnects through the node's CIC. We assume that a security compromise in one node does not mean that the whole cluster system is compromised. This assumption is based on that system administrators can apply appropriate confinement schemes such as logical partitioning, sandboxing, or multi-level security to confine security attacks to predefined resources, We also assume that the switch/router are vulnerable to software attacks, so they can be configured to mirror all traffic to some ports.

Interestingly, under the above assumptions, a physical intruder and a software hacker will do a similar thing; they capture and inject packets from and to the cluster interconnects. Through capturing attacks, they can steal data. If a cluster system deals with confidential or classified data, data theft itself is a serious problem. Besides, captured plaintext packets may reveal information of a

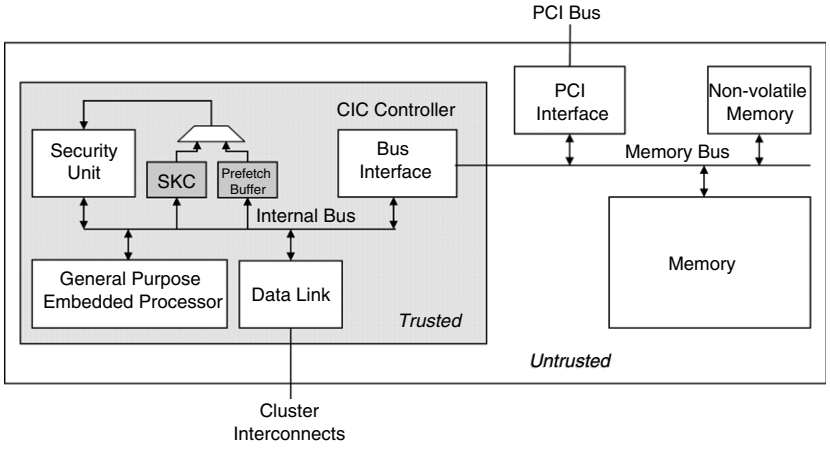


Fig. 1. A CIC design with security function.

global root privilege. Through injecting attacks, they can inject new packets or modify in-transit packets. Without a proper authentication scheme, any illegal modification on confidential data could go unnoticed. In addition, the hacker can stage a spoofing attack using fake source addresses so that it reaches other nodes beyond confined resources.

2.2. Solutions to secure user-level communications

The user-level communication (ULC) has been introduced and adopted in high-performance cluster systems to avoid a long latency with a large host CPU overhead. It allows applications to bypass the OS and access network adapters directly, thus reduces memory copies dramatically. The simplest solution to provide security to ULC is to use a host CPU for encryption and decryption. However, this approach suffers from significant performance overheads by consuming a large amount of CPU time.

Another possible solution is the security coprocessor approach; security coprocessors can take over the host CPU's cryptographic operations. In fact, this approach was thoroughly investigated in [17]. It found out in real experiments of various security coprocessors that the use of the coprocessor needs additional PCI transactions for memory copies to/from the coprocessor and they become a limiting factor of system performance.

By agreeing with [17]'s conclusion that the cryptographic support for secure communication needs to be done in places other than the security coprocessor, we advocate that a CIC augmented with security hardware should implement secure ULC. This CIC approach will not need the additional PCI transactions for memory copies. Applications can simply copy data to/from the CIC as in normal ULC operations so that the CIC takes care of all cryptographic operations by using the security hardware in it. Therefore, we believe that the CIC approach not only satisfies ULC's requirements for low latency and less OS involvement but also efficiently protects the communication messages in cluster interconnects for security.

3. Architectural support for secure cluster communication

In this section, we present a detailed architecture for an SKC with an analysis on its size. Then, we develop an analytical model in order to find the right size of the SKC. For further enhancement, we suggest a coscheduling-aware prefetching scheme.

3.1. Session key cache architecture

In the fine-grained security scheme where a pair of communicating processes (or queue-pairs in InfiniBand) share a unique session key as proposed in [22], each packet needs to carry a session key identifier so that the receiving CIC can look up a corresponding session key. If all the session keys are stored in an off-chip external memory, every packet transmission requires additional delay for accessing the session keys. To minimize average session key access time, we propose the CIC should have an on-chip SRAM, referred to as *session key cache (SKC)*, for storing recently used session keys.

Fig. 1 depicts a general CIC design. The CIC controller is assumed to be tamper-resistant while other components are not. The embedded processor is in charge of all communication processing including packetization, connection management, and protection. The Data Link unit represents any data link layer module that manages data send/receive across physical links by performing error detection, flow control, and buffering. The Security Unit controlled by the embedded processor is a generic module for implementation of any security algorithms.

For security management, each embedded processor needs to be fabricated to carry at least one unique secret key or a private key to protect an initial security setup such as distributing a system-wide secret key. To avoid long system re-initialization at every system reboot, it is better for the CIC to store the system-wide secret key and related secret information in non-volatile memory. Note that it is necessary to encrypt/authenticate all data available in the untrusted area.

The SKC stores recently accessed session keys. Upon a session key cache miss, the SKC fetches the session key from the external memory. Least Recently Used (LRU) is used as the replacement policy to utilize temporal locality. For now, the SKC is assumed to be a fully associative cache in which a session key can be stored at any location. Each cache entry should be large enough to hold one session key and its identifier. To further increase the session key hit rate, we propose to add a small buffer next to the SKC, called a *Prefetch Buffer*, which will be explained in Section 3.3.

To estimate our scheme's impact on power and space of the CIC as well as on the overall performance, we choose the Galois/Counter Mode (GCM) [29] because its authenticated encryption is enough to prevent eavesdropping and authentication attacks described in our threat model. GCM is widely adopted in numerous hardware-based security studies, as in [22,42,43]. Its space and energy consumption is analyzed as follows: a recently designed GCM's throughput is about 35 Gbps at 271 MHz clock rate, consuming around 500,000 gates [43]. According to [16], IBM can make an ASIC with high density (1480 kgates/mm^2) and low power (2.6 nW/MHz/Gate @ 0.9 V) using recent 45 nm technology. By using this technology, one GCM block will consume 0.35 W at 271 MHz clock speed and occupy 0.34 mm^2 space.

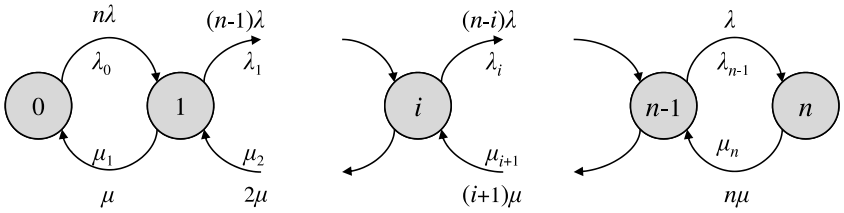


Fig. 2. Markov chain for traffic aggregation of n processes.

Meanwhile, we can estimate the space and power overhead of the SKC and the prefetch buffer by using CACTI model with 45 nm technology. An 16-kbyte SKC with a block size of 32 bytes needs 2.3 mm^2 space and 0.1 W, and a 16-entry prefetch buffer needs 0.19 mm^2 space and 0.09 W. Therefore, the total space and power for a GCM, an SKC, and a prefetch buffer is 2.83 mm^2 and 0.54 W, respectively.

3.2. Size of the SKC

A bigger SKC will increase the SKC hit rate, but its size will soon begin to matter with the big space overhead and long access time as well as hardware cost. In this section, we develop an analytical model to estimate a proper cache size by modeling process-to-process communications in cluster traffic.

One study [41] modeled and analyzed the workload characteristics of NAS parallel benchmarks. It showed that most benchmarks follow a traditional ON–OFF traffic pattern. That is, each process will have a communication period *ON*, followed by a computation period *OFF* that has few communications. The most interesting observation in the study is that the lengths of ON and OFF are well fitted to exponential distributions. Owing to the distribution's memoryless characteristic, we can analytically model how many processes will be in ON stage concurrently. n processes with the ON/OFF traffic pattern can be represented as a Markov chain with state space $\{0, 1, \ldots, n\}$, where n is the number of processors in a cluster system, assuming that one process is running on each processor. Please note that this assumption is a simplification in order to give a more clear explanation. A transition happens only between adjacent states as depicted in Fig. 2. State i means that the number of processes in the ON stage is i. Let λ and μ be the arrival rates of the OFF and ON stages of each process, respectively. An arrival of the OFF stage initiates a start of the ON stage. Since all processes begin at the OFF stage, the transition rate from state 0 to state 1, λ_0, is $n\lambda$. More formally, the transition rate from state i to state $i+1$ is $a_{i,i+1} = \lambda_i = (n-i)\lambda > 0$, and likewise $a_{i+1,i} = \mu_{i+1} = (i+1)\mu > 0$ for $i = 0, 1, \ldots, n$. We obtain the probability of being in state i, P_i, by solving the balance equation $\lambda_i P_i = \mu_{i+1} P_{i+1}$ as shown in Eq. (1).

$$P_i = P_0 \binom{n}{i} \left(\frac{\lambda}{\mu}\right)^i. \tag{1}$$

With Eq. (1) and $\sum_{i=0}^{n} P_i = 1$, the probability of zero active process P_0 can be expressed as

$$P_0 = 1 \Big/ \sum_{i=0}^{n} \binom{n}{i} \left(\frac{\lambda}{\mu}\right)^i. \tag{2}$$

Using these equations, we can get a cumulative distribution function $F_n(x)$ shown in Eq. (3). For any integer x ($0 \le x \le n$), $F_n(x)$ represents the probability that the random variable X is less than or equal to x.

$$F_n(x) = P_n(X \le x) = \sum_{i=0}^{x} P_i = P_0 \sum_{i=0}^{x} \binom{n}{i} \left(\frac{\lambda}{\mu}\right)^i. \tag{3}$$

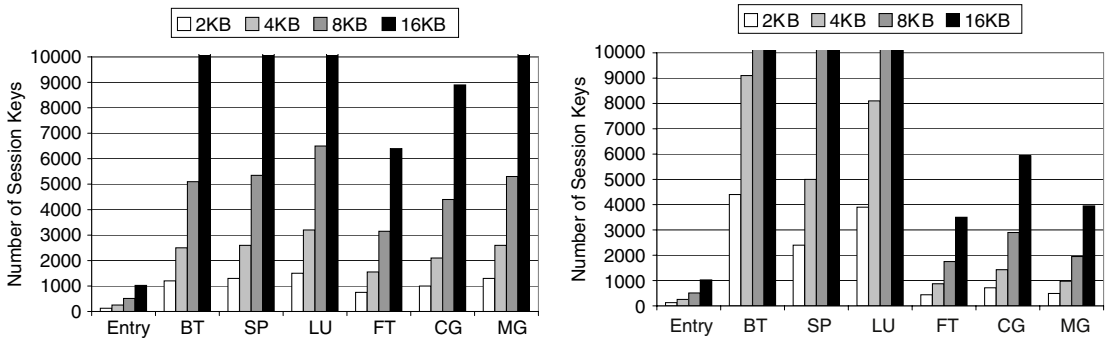
We can use this equation in two ways. First, we can estimate an SKC hit rate p when a cache size x and the number of processes n are given. For example, when $\lambda = \mu$, $x = 3$, and $n = 4$, $P_0 = P_4 = \frac{1}{16}$, $P_1 = P_3 = \frac{1}{4}$, and $P_2 = \frac{3}{8}$. We expect that p will be at least $\frac{15}{16}$ because $\sum_{i=0}^{3} P_i = \frac{15}{16}$. Second, we can also estimate n that a cache of size x supports with a target p. n is referred to as *cache capacity*. Similar to the previous example, if a target p is equal to 0.9, its cache capacity is 4 because $\sum_{i=0}^{3} P_i = \frac{15}{16} > 0.9$, and $\sum_{i=0}^{3} P_i = \frac{13}{16} < 0.9$ when $\lambda = \mu$ and $n = 5$. To CIC architects, the latter usage will be more interesting because they can project the size of a cluster system that a cache size under consideration can support with a target SKC hit rate.

Note that in this model we implicitly assumed that a process communicates with all other processes with an equal probability. [18] reported that some parallel applications show this traffic pattern. However, according to [3], there are also some parallel applications including NAS benchmarks, most processes of which communicate with a small number of processes more frequently than the others. It is noted that a cache provides a higher cache hit rate for a traffic with the locality. Therefore, our assumption of equal distribution is more general to investigate cache capacity.

To estimate cache capacities of real cluster systems, we use an analysis of NAS parallel benchmarks running on an SGI Origin 2000 and an IBM SP-2 system [41]. The analysis presented each benchmark's average ON/OFF length (a reciprocal of the arrival rate). We set p to 0.9 and x to 2, 4, 8, and 16-kbyte caches. Since any two communicating processes share a session key, a cache capacity directly represents the total number of session keys that an SKC can support. Fig. 3 shows cache capacities for those benchmarks. In most cases, the cache capacities are much larger than the number of cache entries represented as *Entry*. For example, in the SGI Origin 2000, the cache capacity of 2KB SKC in BT benchmark is about 1000 while its Entry is 128. This means that in BT benchmark SKC's hit rate will be higher than 90% until there are 1000 session keys. The capacity is quite surprising because the 2KB SKC can hold up to 128 session keys. The reason for this is that the OFF stage duration is at least several times longer than the ON stage duration. That is, the number of processes that are simultaneously active is relatively small. Therefore, we can expect that even a small SKC can support the communication with a fairly large number of processes with a high SKC hit rate. Some might think that there will be few cluster systems that need thousands of sessions keys. However, according to the supercomputer Top 500 list released in November 2007, the percentage of supercomputers that have more than one thousand processors is 91%. Considering that its percentage in November 2005 list was only 29%, the number of processors in cluster systems will continue to increase quickly. Therefore, the large cache capacities supported by a small SKC will be practical.

3.3. Coscheduling-aware prefetching scheme

In order to maximize the cluster system's overall throughput, researchers have proposed coschedulers that can schedule physically distributed processes of a job to run as concurrently as possible [1,4,32,39]. If these coschedulers are used in cluster systems,



(a) SGI Origin 2000. (b) IBM SP-2.

Fig. 3. Cache capacities of SKC for NAS benchmarks for target hit rate 0.9.

multiple processes are running on each cluster node and there will be context switches inevitably. The problem of context switches is that SKC hit rates will decrease because a newly scheduled process needs to use a completely different set of session keys for communicating with whole different nodes. As a result, many subsequent session key accesses for the new process will miss until its SKC warms up. The best solution to this problem would be to synchronize the contents of SKC according to scheduling decisions of the coscheduler. That is, when a coscheduler picks a process to run next, the OS provides the CIC with the decision so that the CIC can prefetch session keys for that process. However, it will cause OS interventions and additional communications between the OS and the CIC. Moreover, it will increase the complexity of the CIC design for the synchronization. To solve this problem, we propose a simple prefetching scheme that can synchronize SKC and coscheduling decisions without OS interventions.

Before describing the prefetching scheme, we would like to begin with explaining why coschedulers are proposed and how they work. A drawback of the batch scheduler that assigns a number of processors to an application exclusively is the low CPU utilization because a CPU often waits for I/O operations or messages. A simple remedy is to assign multiple applications to each node by using a local scheduler. However, since it does not consider the behavior of other processes running on other nodes, one process would have to wait until the other process wakes up, resulting in the low CPU utilization. In contrast, the gang scheduler can explicitly run all processes of a job at the same time [10,14], but unfortunately it becomes quite complex in large-scale cluster systems. To remove the above mentioned problems, researchers proposed several communication-driven coscheduling schemes such as SB, DCS-SB, and PB. These scheduling schemes are generally called *implicit coscheduling*, because a message arrival from a process in a remote node implicitly notices that the process is running actively on the node. Its main benefit is that it still produces better CPU utilization and throughput than the local scheduler, while it does not need a complex global control mechanism as in the gang scheduler [1,32,39]. In the SB [1], after a process sends a message, it spins (or busy wait) for a fixed time before blocking itself, hoping that a reply message arrives within the spinning time. The DCS-SB [32,39] acts like the SB except that, when a message arrives at a blocked process, it increases the priority of the process. The PB [32] periodically boosts the priority of processes that have unconsumed messages in a round-robin fashion.

Our prefetching scheme takes advantage of the fact that the aforementioned implicit coschedulers use message arrivals as a hint for coscheduling. In other words, if a message arrives at a process and the process is not occupying its CPU currently, the CIC presumes that implicit schedulers will try to schedule the process soon. Based on this, the CIC loads the process's frequently used session keys to a prefetch buffer. When a new message arrives, its session key can be found in the SKC if the message belongs to the current process, or can be found in the prefetch buffer if the message is for the new process. For this, a session key should be looked up in the SKC and the prefetch buffer in parallel. If its session is found in the prefetch buffer, the session key is moved to the SKC and one session key needs to be evicted from the SKC. At every session key eviction, the CIC updates the list of frequently accessed session keys for the process that the evicted session key belongs to. This operation can be done by the embedded processor later, so the operation is not in the critical path.

4. Performance evaluation

4.1. Simulation platform

Fig. 4 depicts the simulation platform for the coscheduled cluster communications. Shaded rectangles represent what we developed, modified, or executed in this study. We use six NAS parallel benchmarks [33] that are widely used for measuring the system performance in cluster/supercomputing environments. To measure the behavior of a commercial benchmark, we also use SPECjbb2000 [40] that models the workload of Java application servers. In order to capture traces for five NAS benchmarks, CG, MG, LU, SP and BT, we first modified source codes of the benchmarks to print all packets' source and destination with a timestamp. We ran the modified benchmarks on an SGI Altix 3700 consisting of 128 Itanium-2 processors and 256-Gbyte main memory with a batch scheduler where each processor is assigned to only one program at a time. Due to our network simulator's limited scalability that is inevitable to provide cycle-accuracy, we used the small problem size (Class A) of the NAS benchmarks using 16 processors. For FT and SPECjbb2000 benchmarks, we used Simics full-system simulator [25] with GEMS [27] modules. We modified source codes of GEMS to capture traces when packets are injected to the interconnection network.

Our cluster network simulator used in [19] includes packet-level switches and CICs that are compliant with InfiniBand, one of the promising cluster interconnects. To simulate more realistic cluster communications, we made three major changes to the simulator: timed simulation, coscheduling simulation, and cache simulation. First, we developed a trace-driven cluster simulator by modifying the existing cluster network simulator in which all packets were generated randomly with a certain distribution. An important requirement of the new simulator was that packet injection time needs to be adjusted dynamically. This is because a packet may come across a shorter or longer network path during simulation due to differences between the simulated networks and the real networks. To apply these dynamics to simulation, our simulator provides the relative-timed injection by maintaining the actual time distance between two consecutive packets' timestamps.

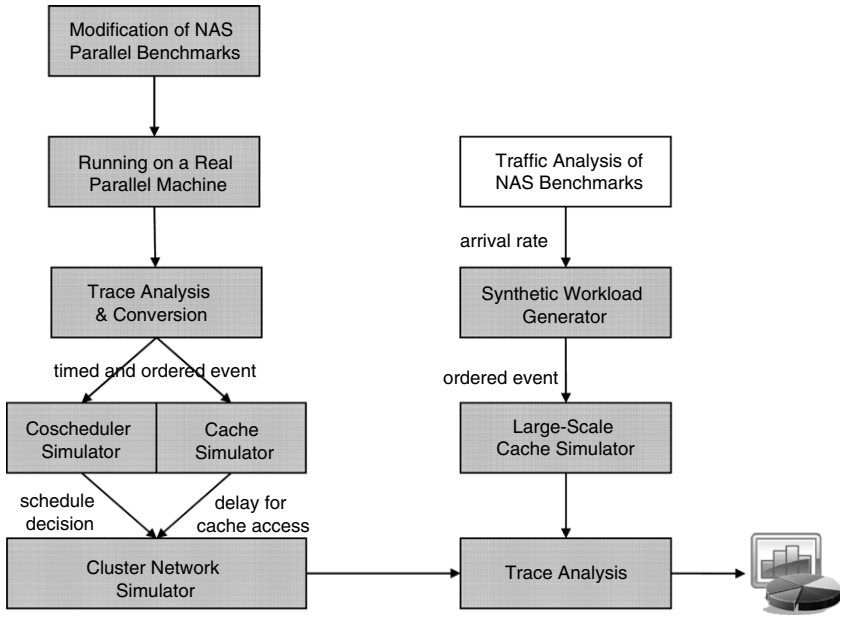


Fig. 4. Coscheduled cluster communication simulation platform.

Table 1
Original and simulated execution time of NAS benchmarks.

Time (s)	CG	MG	LU	SP	BT
Original	3.72	11.86	268.61	244.12	291.73
Simulation	3.85	12.80	257.85	233.90	278.40

In other words, let us suppose that in a real execution a node receives and sends two packets at t and $t + i$, respectively. If the first event (packet arrival) occurs at t' during simulation, then the second event (packet injection) should be scheduled at $t' + i$. This timed injection prevents unrealistically early or late packet injections. Because of this effect, the simulated execution time of the benchmarks is different from the actual execution time in the real system as shown in Table 1.

Second, to simulate a cluster system in which multiple applications are competing for network and computing resources, we implemented a Linux local scheduler and four coschedulers: SB, DCS-SB, PB, and gang. The Linux local scheduler is the same as the one in the Linux kernel 2.6. The simulation parameters of each coscheduler are the following: the spinning time for SB and DCS-SB is 750 μs, a regular priority boost in PB occurs every 2 ms, and the interval of the gang scheduling is 20 ms. All the parameters are optimized through multiple simulations, and other implementation issues are described in [4,32].

Third, to simulate the SKC, we also implemented a cache simulator in the CIC. Before sending and after receiving a packet, the CIC checks its SKC to see whether the necessary session key is cached. If found, it takes only a cache access time to read the session key, but if not, a memory access time will be consumed. In Fig. 4, we draw the three simulators (coscheduler simulator, cache simulator, and cluster network simulator) separately for better understanding, but in fact they are integrated into the cluster network simulator.

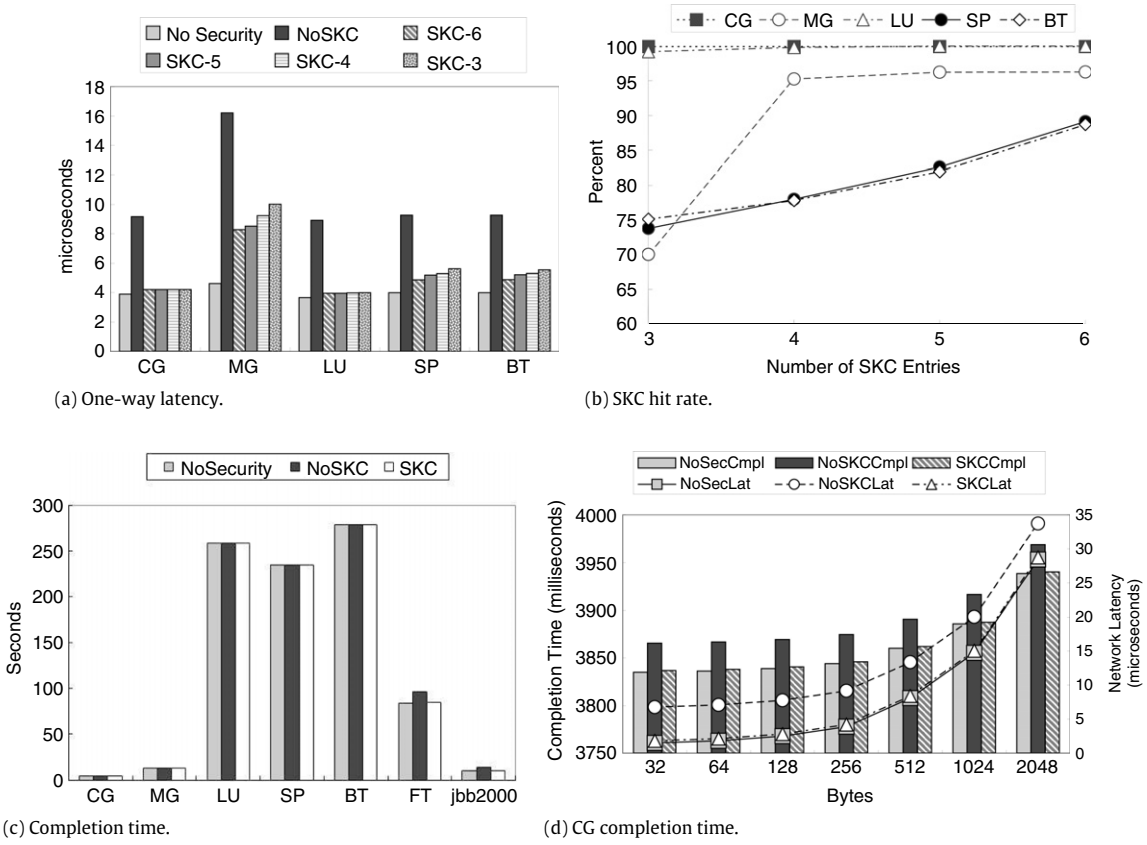
Additionally, we developed a synthetic packet generator and a cache simulator to simulate the analytical ON/OFF traffic model of benchmarks explained in Section 3.2. Note that, since we are interested in the SKC hit rate, we use only the sequence of packet arrivals, not the interval between packet arrivals. The benefit of this simulation is to overcome the time limitation in simulating a large number of long-running processes in a short simulation time, which would be impossible in a cycle-accurate simulator.

Finally, there are several important simulation parameters. A GCM implementation presented in [43] requires 12 cycles to encrypt/decrypt the first 128 bit block and one cycle for each additional block with last two cycles to generate an authentication tag. If a 256 byte packet is in use, the additional delay is 30 cycles in sending or receiving a packet. Since this delay needs to occur both in a sender and in a receiver, the total additional one-way delay is 60 cycles that are converted to approximately 220 ns with 270 MHz. According to the CACTI model, the cache access time of a 16 Kbyte fully associative cache is 3 ns. An access time to an external memory in a CIC is set to 2.5 μs, as measured in [30].

4.2. Effectiveness of SKC

Fig. 5(a) compares the average network latencies of several configurations of NAS benchmarks. The reason why we compare the network latencies is that, since all security operations are done in the CIC, the security overhead will be reflected in the network latency. *NoSecurity* represents the original configuration without a security function enabled. *NoSKC* represents the configuration where encryption/decryption and authentication are enabled but the SKC does not exist. *SKC-n* configurations have *n* entries in the SKC. Compared with *NoSecurity*, *NoSKC* has around 160% overhead on the network latency because each packet requires an external memory access for retrieving a session key. In contrast, *SKC-6* reduces the overhead by 50% on average from *NoSKC*, adding 29% overhead to *NoSecurity* configuration. The SKC hit rates of CG and LU shown in Fig. 5(b) are almost 100%, showing little difference in the network latencies of *SKC-n* configurations in Fig. 5(a). However, other benchmarks have relatively low hit rates in small SKCs, resulting in longer network latency. Therefore, we can say that the higher SKC hit rate is positively related to the shorter network latency.

Fig. 5(c) compares the benchmark completion time of the above configurations using NAS and SPECjbb2000 benchmarks. In contrast to our expectations, it does not seem that the big difference in the network latency has a big impact on the overall application completion time except for FT and jbb2000. The similar trend was reported by [28], explaining that many cluster applications are insensitive to the network latency, even though some are sensitive. This implies that NAS parallel benchmarks are not very sensitive



(a) One-way latency.

(b) SKC hit rate.

(c) Completion time.

(d) CG completion time.

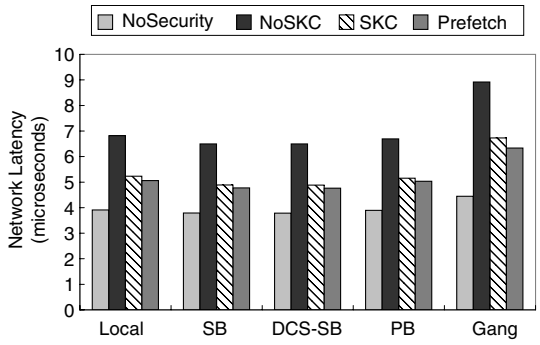
Fig. 5. Performance comparisons of network latency of single application execution.

to the network latency in general. However, we believe that there are some applications that can get a great benefit from the short network latency. For example, real-time multimedia applications used in a remote surgery need quick responses as well as strong security on each packet. Considering that cluster systems are being deployed for diverse applications, it will be useful to minimize the additional network latency while providing fine-grained security. Fig. 5(d) shows the completion time and the network latency of CG with various packet lengths in millisecond-scale. Since the latency is measured till the last flit of a packet arrives at its destination, the average latency also increases as the length of a packet increases. Note that the network latency of SKC is similar to that of NoSecurity, meaning that the additional latency from SKC will be little. This benefit becomes more conspicuous in shorter packets. Without SKC, the additional overhead in a 32 byte packet could be around 180%, as compared to only 28% in a 2048 byte packet.

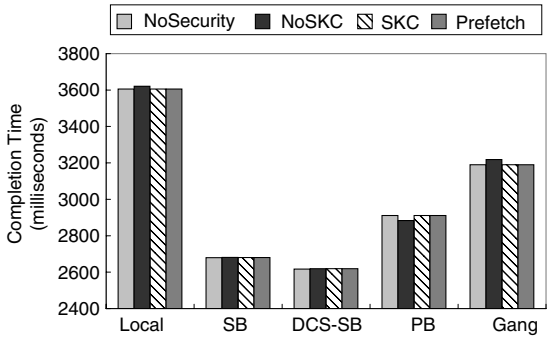
4.3. Effectiveness of prefetching

To make workloads for the coscheduling environment, we assign a mix of NAS and SPECjbb2000 benchmark traces to each node so that a scheduler of the node coschedules the traces. Fig. 6 shows the overall performance of all coschedulers in terms of network latency and completion time. In Fig. 6(a) and (b), each workload uses 15 benchmark traces: three instances per each of the five benchmark. In Fig. 6(c) and (d), 6 traces are used: three instances per each of FT and jbb2000. The number of entries in the SKC is 10, and the prefetch buffer can hold eight session keys. In Fig. 6(a) and (c), *Prefetch* reduces network latency by 5% on average. This performance gain mainly comes from around a 13% hit rate increase in *Prefetch*. However, it is still hard to tell that the shorter latency always enhances the performance in terms of total completion time based on our simulation results shown in Fig. 6(b) and (d). Among coschedulers, DCS-SB and SB have the shortest completion time, and local the longest, which is similar to the results in other literature [4,32]. In the same coscheduler, most completion times are similar except for PB; contrary to our expectation, *NoSKC*'s completion time is even shorter than completion times of other configurations. The main reason for this is that even a subtle difference in the network latency changes the scheduling order of a node, and this change has a chain effect, thus resulting in a completely different scheduling order in the whole cluster system. In our simulation environment, the variability of scheduling orders seems bigger than that of the performance gain from the prefetching scheme, especially in terms of completion time. However, we believe that the shorter network latency will help improve the overall performance in the long term.

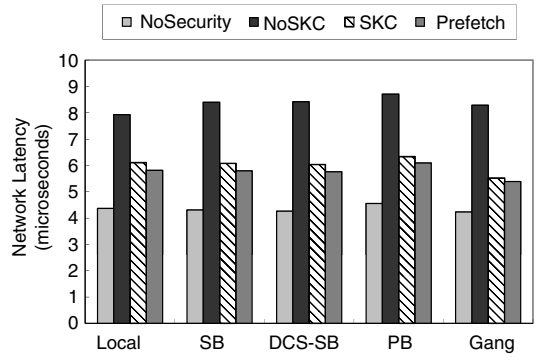
Fig. 7 describes how the network latency and the SKC hit rate are affected by the changes in the size of the SKC and the prefetch buffer. The SKC hit rate encompasses all hits both on the SKC and on the prefetch buffer. In Fig. 7(a), we fix the size of the prefetch buffer to two, and increase the size of SKC. DCS-SB is used as the coscheduler in this simulation. We compare two cases: one with a prefetch buffer (/w) and the other without it (w/o). As expected, the latency decreases and the SKC hit rate increases with the increasing size of SKC. Note that, with a bigger size of the SKC, the additional gain from the prefetch buffer is slightly decreased. This is because more session keys residing in a big SKC yields more hits on the SKC, consequently reducing hits on the prefetch buffer. Therefore, the prefetch buffer is more useful when the size of the SKC is small. Fig. 7(b) depicts how the SKC performance varies as the size of the prefetch buffer increases, with two different sizes of SKC, 5 and 20. In the case of a 20-entry SKC, although the size of the prefetch buffer becomes big enough to contain more sets of prefetched sessions keys of previous SKC misses, the effect of old prefetched



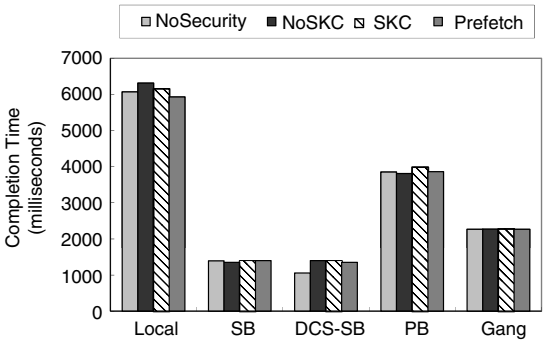
(a) Network latency (CG/MG/LU/SP/BT).



(b) Completion time (CG/MG/LU/SP/BT).

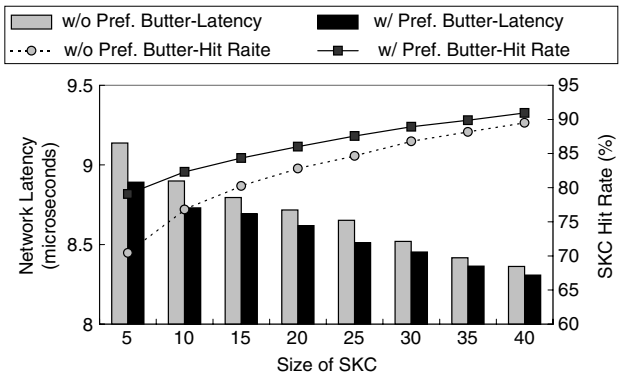


(c) Network latency (FT/jbb2000).

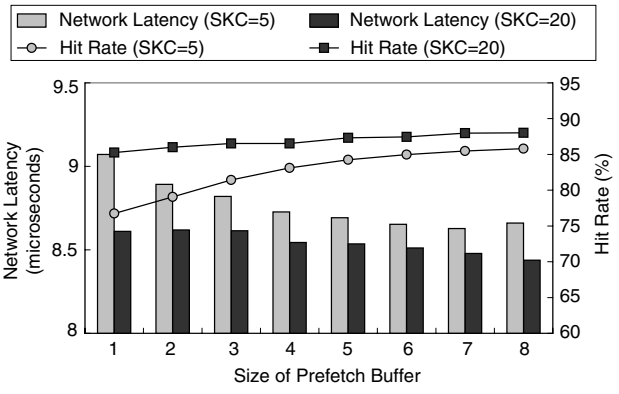


(d) Completion time (FT/jbb2000).

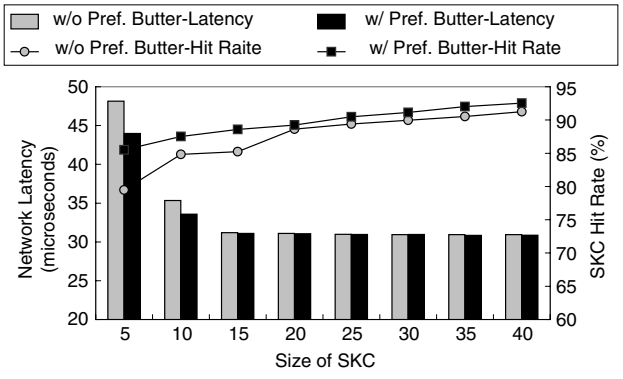
Fig. 6. Performance gain from prefetch buffer on various coschedulers.



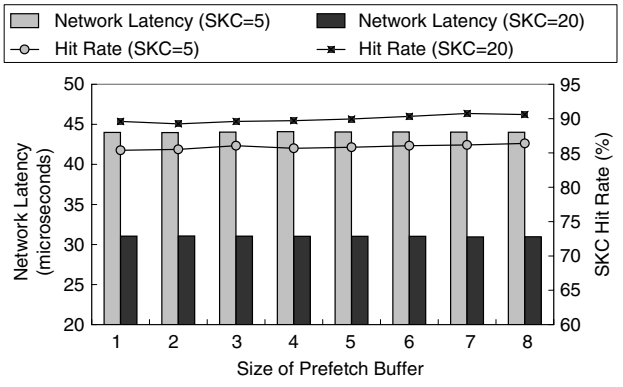
(a) Pref. buf. = 2 (DCS-SB).



(b) SKC = 5,20 (DCS-SB).

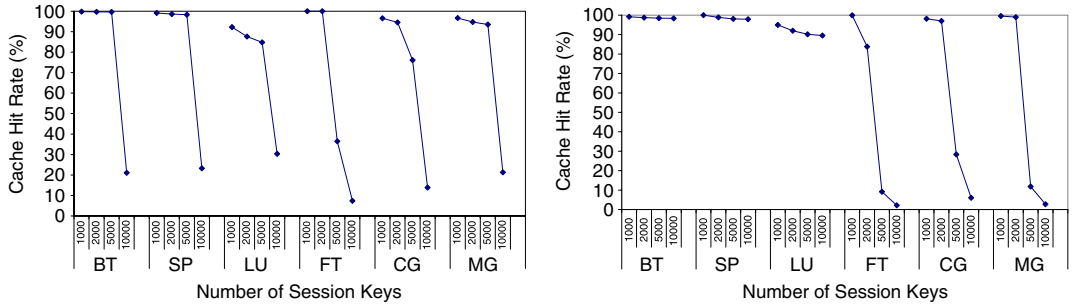


(c) Pref. buf. = 2, (Gang).



(d) SKC = 5,20 (Gang).

Fig. 7. Performance comparison on varying size of SKC and prefetch buffer.



(a) SGI Origin 2000.

(b) IBM SP-2.

Fig. 8. Cache hit rate of NAS benchmarks on a 16 KByte SKC.

session keys is diminishing gradually. In the figure, although the prefetch buffer size increases by eight times from 1 to 8, there is a small performance gain, 0.17 μs in the latency and 2.8% in the SKC hit rate. All other schedulers except the gang scheduler showed the similar trend, so they are omitted in this paper.

Fig. 7(c) and (d) show the gang scheduler's results. To our surprise, as the size of the SKC reaches around 15, neither a prefetch buffer nor a larger SKC improves network latency. This is related to how the gang scheduler coschedules multiple applications. The gang scheduler usually makes all processes of an application in multiple nodes run concurrently, and then it moves to the next application after a fixed time-slice. Since all applications are executed in order, the session keys used by the previous application will not be used for a long time. In other words, once the size of the SKC becomes large enough to hold all session keys of an application, its hit rate will not increase much. This is why the increase in the SKC hit rate in Fig. 7(c) is minimal. For the same reason, larger prefetch buffers make little difference in the network latency and the SKC hit rate as shown in Fig. 7(d). From the above results, we conclude that the effect of the prefetch buffer varies according to the size of SKC and coscheduling policy.

4.4. SKC size

To simulate the SKC behavior on a large-scale cluster system, the synthetic packet generator increases the total number of session keys in one CIC from 1000 to 10,000. Although it is quite improbable for one process to communicate with 10,000 processes at the same time, this simulation can show how a small SKC withstands such extreme loads. The communication pattern between two processes follows the ON/OFF model as described in [41]. In Fig. 8(a) and (b), the 16 Kbyte SKC shows relatively high hit rates in most benchmarks, except for FT in SGI Origin 2000, and FT, CG, and MG in IBM SP-2. The reason for the exceptions is that the cache capacities of the benchmarks are less than 10,000 session keys as shown in Fig. 3. It is noteworthy to point out that LU has a little lower hit rate, even with a large cache capacity in Fig. 3. The reason for this is that LU has 10–100 times shorter ON and OFF durations than those of other benchmarks, which means that communicating processes are alternating frequently, thus resulting in a lower SKC hit rate. Still, we can say that a high cache capacity is strongly and positively related to a high SKC hit rate.

Finally we compare the fully associative SKC with other n-way associative SKC. Although the fully associative cache generally shows the higher hit rate, it has several drawbacks such as long access time, large space, and complex control logic. We tested seven set associative SKCs: 1, 2, 4, 8, 16, 32-ways, and fully associative. We use the simulated BT and LU traffic from Origin 2000. As shown in Fig. 9, the lower associative cache shows a lower cache hit rate in both applications, but 16-way and 32-way caches work comparably to the fully associative SKC. Since this trend is observed in all of the applications we tested, we expect that 16-way or 32-way associativity will be appropriate for general applications in large-scale cluster systems.

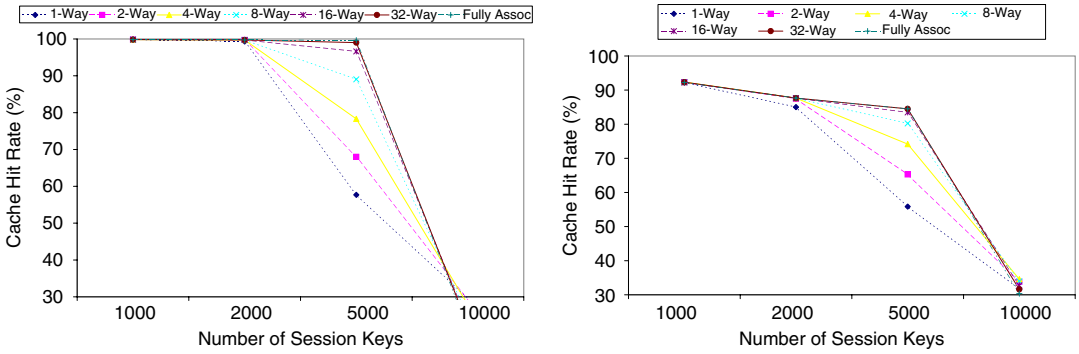
5. Related work

There has been some research to improve security inside clusters. An early study [8] presented security enhancement methods for system area networks. [7] showed that encryption can be used in clusters with a minimal performance impact. [34] proposed the distributed security infrastructure (DSI) to supports a fine-grained cluster-wide security enforcement by providing a process-level resource and access control. [44] identified security characteristics unique to clusters. NVisionCC [20] was presented to enhance the security inside cluster systems by monitoring processes across cluster nodes. [23] proposed an instant attack stopper (IAS) scheme to instantly block inside attackers in InfiniBand cluster systems. Recently, [22] pointed out security vulnerabilities of InfiniBand and proposed a security framework that can be easily integrated into its specifications.

Some recent work has focused on the performance gain by use of the security coprocessor on real systems. [31] analyzed the relative cost of network security protocols, and advocated that the dedicated cryptographic hardware can make any security protocols practically viable. [5] investigated the performance overhead of TLS web servers using the host processor and the security coprocessor approaches. Due to the limited performance gain by the security coprocessor, they concluded that an additional host processor would be more beneficial than the security coprocessor. [17] presented an OpenBSD cryptographic framework. It contains a cryptographic API to provide a uniform access to various cryptographic hardware, thus enabling the high utilization of the hardware. [37] found that cryptographic hardware attached to I/O bus would not improve security performance due to the severe bus contention, and then the authors proposed a scheme similar to the CIC approach, but they left out detailed performance evaluation. Several network adapters equipped with cryptographic hardware are available in the market [6,2]. However, since they were not developed for the ULC of cluster systems, and their detailed architecture is not publicly available, it is difficult to directly compare them with our approach.

6. Conclusions

This paper proposes an architectural support to enable fine-grained secure communications in coscheduled cluster systems with a low performance overhead. We propose a small cache, referred to as SKC, inside the cluser interconect controller to store recently accessed session keys to support fine-grained security. To further increase hit rate of the cache, we propose a prefetch buffer to fetch session keys from memory ahead of time by predicting the



(a) BT in SGI Origin 2000.

(b) LU in IBM SP-2.

Fig. 9. Cache associativity of a 16-KByte SKC.

job scheduler's decision. In addition, by constructing an analytical model, we are able to estimate the number of concurrent processes that a given size of SKC can support.

The important conclusions of this work are the followings. First, we advocate that it is very desirable for the cluster interconnect controller to perform security operations, because this approach incurs virtually no performance overhead to the host CPU and no additional PCI transactions, while retaining the performance benefit of the user-level communication through bypassing the host CPU. Second, when the communication pattern of a cluster application shows a high temporal locality, most of session key accesses will be found in the SKC, thus incurring little performance overhead in managing the fine-grained session keys. Simulation results show that network latency is reduced by 50% on average, compared with non-SKC configurations. Third, if the job scheduling decision is predictable, session keys of the job can be fetched into the prefetch buffer to get the session keys ready when they are needed. We implemented the Linux local scheduler and four coschedulers including SB, DCS-SB, PB, and gang scheduler, and then we developed their prediction schemes. Simulation results show that the prefetch buffer reduces network latency by 5% on average from that of SKC-only configurations. Fourth, based on the NAS parallel benchmarks' traffic characteristics, the analytical model shows that even a 16-Kbyte SKC yields a high hit rate even when managing thousands of session keys.

We are currently examining two possible extensions to this work. First, we plan to implement the proposed architecture on a reconfigurable NIC. Second, we are exploring other design alternatives to incorporate more security features in the cluster interconnect controller such as fast attack detection and response. Finally, we will investigate a general communication model that reflects a more realistic application behavior.

References

[1] A.C. Arpaci-Dusseau, D.E. Culler, A.M. Mainwaring, Scheduling with implicit information in distributed systems, in: Proceedings of ACM SIGMETRICS '98, ACM Press, 1998, pp. 233–243.

[2] Broadcom, Broadcom gigabit ethernet controller. http://www.broadcom.com/products/Small-Medium-Business/Gigabit-Ethernet-Controllers/BCM5752M.

[3] S. Chodnekar, V. Srinivasan, A.S. Vaidya, A. Sivasubramaniam, C.R. Das, Towards a Communication Characterization Methodology for Parallel Applications, vol. 0, IEEE Computer Society, Los Alamitos, CA, USA, 1997, p. 310.

[4] G.S. Choi, J.-H. Kim, D. Ersoz, A.B. Yoo, C.R. Das, Coscheduling in clusters: Is it a viable alternative? in: Proceedings of the 2004 ACM/IEEE Conference on Supercomputing (SC '04), IEEE Computer Society, Washington, DC, USA, 2004, p. 16.

[5] C. Coarfa, P. Druschel, D.S. Wallach, Performance analysis of TLS web servers, ACM Transactions on Computer Systems 24 (1) (2006) 39–69.

[6] 3Com, 3Com Secure NIC. http://www.3com.com/products/en_US/detail.jsp?tab=features&pathtype=purchase&sku=3CR990B-97.

[7] K. Connelly, A.A. Chien, Breaking the barriers: High performance security for high performance computing, in: Proceedings of the 2002 Workshop on New Security Paradigms, ACM Press, New York, NY, USA, 2002, pp. 36–42.

[8] R. Dimitrov, M. Gleeson, Challenges and new technologies for addressing security in high performance distributed environments, in: Proceedings of the 21st National Information Systems Security Conference, 1998, pp. 457–468.

[9] J.G. Dyer, M. Lindemann, R. Perez, R. Sailer, L. van Doorn, S.W. Smith, S. Weingart, Building the IBM 4758 secure coprocessor, Computer 34 (10) (2001) 57–66.

[10] Y. Etsion, D.G. Feitelson, User-Level Communication in a System with Gang Scheduling, Parallel and Distributed Processing Symposium, International 1 (2001) 10058a.

[11] B. Gassend, G.E. Suh, D. Clarke, M. van Dijk, S. Devadas, Caches and hash trees for efficient memory integrity verification, in: Proceedings of the 9th International Symposium on High-Performance Computer Architecture, IEEE Computer Society, Washington, DC, USA, 2003, p. 295.

[12] P. Gutmann, An open-source cryptographic coprocessor, in: Proceedings of the 9th Conference on USENIX Security Symposium, USENIX Association, Berkeley, CA, USA, 2000, pp. 8–8.

[13] Hackers Hit Supercomputing Giants, Associated Press, 2004, April 19.

[14] A. Hori, H. Tezuka, Y. Ishikawa, Highly efficient gang scheduling implementation, in: Proceedings of the 1998 ACM/IEEE Conference on Supercomputing, IEEE Computer Society, Washington, DC, USA, 1998, pp. 1–14.

[15] HPC wire, InfiniBand Cluster Deployed at SC05. November 12–18, 2005 http://news.taborcommunications.com/msgget.jsp?mid=506904.

[16] IBM AISC solutions. http://www-01.ibm.com/chips/techlib/techlib.nsf/techdocs/9bf3ec138a82223b8725708b004c6471/$file/asicbrochure_june0407_final.pdf.

[17] A.D. Keromytis, J.L. Wright, T.D. Raadt, M. Burnside, Cryptography as an operating system service: A case study, ACM Transactions on Computer Systems 24 (1) (2006) 1–38.

[18] J. Kim, D.J. Lilja, Characterization of communication patterns in message-passing parallel scientific application programs, in: Proceedings of the Second International Workshop on Network-Based Parallel Computing, Springer-Verlag, London, UK, 1998, pp. 202–216.

[19] E.J. Kim, K.H. Yum, C.R. Das, M. Yousif, J. Duato, Performance enhancement techniques for InfiniBand architecture, in: Proceedings of the 9th International Symposium on High-Performance Computer Architecture, IEEE Computer Society, Washington, DC, USA, 2003, p. 253.

[20] G. Koenig, X. Meng, A. Lee, M. Treaster, N. Kiyanclar, W. Yurcik, Cluster security with NVisionCC: Process monitoring by leveraging emergent properties, in: IEEE International Symposium on Cluster Computing and the Grid, 2005. CCGrid 2005. vol.1, 2005, pp. 121–132.

[21] J. Krim, R.O. Jr., Data Under Siege, The Washington Post, http://www.washingtonpost.com/wp-dyn/articles/A19982-2005Mar9.html (March 10, 2005).

[22] M. Lee, E.J. Kim, A comprehensive framework for enhancing security in infiniband architecture, IEEE Transactions on Parallel and Distributed Systems 18 (10) (2007) 1393–1406.

[23] M. Lee, E.J. Kim, K.H. Yum, M.S. Yousif, Instant attack stopper in infiniband architecture, in: CCGRID '05: Proceedings of the Fifth IEEE International Symposium on Cluster Computing and the Grid (CCGrid'05) - vol.1, IEEE Computer Society, Washington, DC, USA, 2005, pp. 105–110.

[24] D. Lie, C. Thekkath, M. Mitchell, P. Lincoln, D. Boneh, J. Mitchell, M. Horowitz, Architectural support for copy and tamper resistant software, SIGPLAN Notices 35 (11) (2000) 168–177.

[25] P.S. Magnusson, et al., Simics: A full system simulation platform, IEEE Computer 35 (2) (2002) 50–58.

[26] G. Markowsky, L. Markowsky, Survey of supercomputer cluster security issues, in: Security and Management, 2007, pp. 474–480.

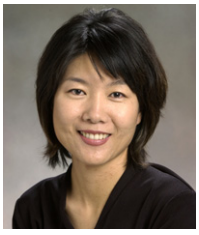
[27] M.M.K. Martin, D.J. Sorin, B.M. Beckmann, M.R. Marty, M. Xu, A.R. Alameldeen, K.E. Moore, M.D. Hill, D.A. Wood, Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset, SIGARCH Computer Architecture News 33 (4) (2005) 92–99.

[28] R.P. Martin, A.M. Vahdat, D.E. Culler, T.E. Anderson, Effects of communication latency, overhead, and bandwidth in a cluster architecture, in: Proceedings of the 24th Annual International Symposium on Computer Architecture, ISCA '97, ACM Press, New York, NY, USA, 1997, pp. 85–97.

[29] D. McGrew, J. Viega, The Galois/Counter Mode of Operation (GCM), Submission to NIST Modes of Operation Process (2004).
[30] Mellanox, InfiniHost III ex memfree mode performance. http://www.mellanox.com/pdf/whitepapers/PCIxVsMemfree_WP_100.pdf.
[31] S. Miltchev, S. Ioannidis, A.D. Keromytis, A study of the relative costs of network security protocols, in: Proceedings of the FREENIX Track: 2002 USENIX Annual Technical Conference, USENIX Association, Berkeley, CA, USA, 2002, pp. 41–48.
[32] S. Nagar, A. Banerjee, A. Sivasubramaniam, C.R. Das, Alternatives to coscheduling a network of workstations, Journal of Parallel and Distributed Computing 59 (2) (1999) 302–327.
[33] NAS parallel benchmarks. http://www.nas.nasa.gov/Software/NPB/.
[34] M. Pourzandi, A new distributed security model for Linux clusters, in: ATEC '04: Proceedings of the Annual Conference on USENIX Annual Technical Conference, USENIX Association, Berkeley, CA, USA, 2004, pp. 43–43.
[35] W. Shi, H.-H.S. Lee, M. Ghosh, C. Lu, A. Boldyreva, High efficiency counter mode security architecture via prediction and precomputation, in: Proceedings of the 32nd Annual International Symposium on Computer Architecture, IEEE Computer Society, Washington, DC, USA, 2005, pp. 14–24.
[36] S.W. Smith, V. Austel, Trusting trusted hardware: Towards a formal model for programmable secure coprocessors, in: Proceedings of the 3rd conference on USENIX Workshop on Electronic Commerce, USENIX Association, Berkeley, CA, USA, 1998, pp. 8–8.
[37] J. Smith, C. Traw, D. Farber, Cryptographic support for a Gigabit network, Proceedings of INET (1992) 229–237.
[38] S.W. Smith, S. Weingart, Building a high-performance, programmable secure coprocessor, Computer Networks 31 (9) (1999) 831–860.
[39] P. Sobalvarro, W.E. Weihl, Demand-based coscheduling of parallel jobs on multiprogrammed multiprocessors, in: Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing, Springer-Verlag, London, UK, 1995, pp. 106–126.
[40] SPECjbb2000 benchmark suite. http://www.spec.org/jbb2000/.
[41] A. Waheed, J. Yan, Workload characterization of cfd applications using partial differential equation solvers, in: Proceedings of Workshop on Workload Characterisatisation in High-Performance Computing Environments, Montreal, Canada, 1998.
[42] C. Yan, D. Englender, M. Prvulovic, B. Rogers, Y. Solihin, Improving cost, performance, and security of memory encryption and authentication, in: Proceedings of the 33rd Annual International Symposium on Computer Architecture, IEEE Computer Society, Washington, DC, USA, 2006, pp. 179–190.
[43] B. Yang, S. Mishra, R. Karri, A high speed architecture for Galois/Counter Mode of Operation (GCM), Cryptology ePrint Archive, Report 2005/146 (2005).
[44] W. Yurcik, G.A. Koenig, X. Meng, J. Greenseid, Cluster security as a unique problem with emergent properties: Issues and techniques, in: Proceedings of Linux Revolution, 2004, (CDROM).

Manhee Lee received the BE and ME degrees from the Department of Computer Engineering, Kyungpook National University, Korea, in 1995 and 1997, respectively, and the Ph.D. Degree from the Computer Science Department, Texas A&M University in 2008. He is currently a hardware engineer at Cisco systems. He worked as a network researcher at the Korea Institute of Science and Technology Information for seven years. His research interests include computer architecture, parallel/distributed systems, secure computing architecture, secure cluster computing, Internet security, and computer networks.

Baik Song An received the BS and MS degrees of computer science from Seoul National University, Korea, in 1999 and 2001, respectively. Currently he is a Ph.D. student in the Department of Computer Science and Engineering, Texas A&M University. He worked as a member of engineering staff in Electronics and Tel ecommunications Research Institute in Korea from 2001 to 2006. His research interests include computer architecture, parallel/distributed systems, secure processor architecture, operating system, system software and embedded systems.



Eun Jung Kim received the BS degree in computer science from the Korea Advanced Institute of Science and Technology, Korea, in 1989, the MS degree in computer science from the Pohang University of Science and Technology, Korea, in 1994, and the Ph.D. Degree in computer science and engineering from the Pennsylvania State University in 2003. She is an assistant professor in the Department of Computer Science, Texas A&M University. From 1994 to 1997, she worked as a member of the technical staff in the Korea Telecom Research and Development Group. Her research interests include computer architecture, parallel/distributed systems, low-power design, secure computing, performance evaluation, and fault tolerant computing. She is a member of the IEEE Computer Society and the ACM.