단일 프로세스 공간을 제공하는 클러스터 시스템의 설계 및 구현

최 민° 이대우 박동근 유정록 맹승렬
한국과학기술원 전자전산학과 전산학전공
{mchoi°, dwlee, dgpark, jlyu, maeng}@camars.kaist.ac.kr

Design and Implementation of Clusters with Single Process Space

Min Choi° DaeWoo Lee, DongGun Park, JungLok Yu, SeungRyoul Maeng
Division of Computer Science, Department of Electrical Engineering and Computer Science
Korea Advanced Institute of Science and Technology

요 약

Single system image(SSI) have been the mainstay of high-performance computing for many years. SSI requires the integration and aggregation of all types of resources in a cluster to present a single interface to users. In this paper, we describe a cluster computing architecture with the concept of single process space(SPS) where all processes share a uniform process identification scheme. With SPS, a process on any node can create child process on the same or different node or communicate with any other process on a remote node, as if they are on a single node. For this purpose, SPS is built with the support of unique cluster-wide pid, signal forwarding, and remote fork. We propose a novel design of SPS cluster which addresses the scalability and flexibility problem of traditional clusterwidely unique pid implmentation by using blocked pid assignment. We have implemented this new design of SPS cluster, and we demonstrate its performance by comparing it to Beowulf distributed process space. Benchmark performance results show that our design of SPS cluster realized both scalability and flexibility that are essential to building SPS cluster.

1. Introduction

Recently, the rapid progress of network and microprocessor technologies has made cluster computing systems an attractive alternative to massively parallel machines [1]. Workstation clusters can share computing resources more easily, and node capability can easily be increased by adding commodity hardware. For this reason, the usage of cluster systems is becoming increasingly widespread. In fact, 93 of the top 500 supercomputer sites are applications of a cluster system [2]. The increasing popularity requires the cluster systems with better utilization in terms of convenience, performance, scalability, and reliability. Thus, we have to provide a single system image(SSI) view of a cluster to users or administrators.

SSI enables a cluster of PCs or workstations to be used as a single computing unit in an efficient and scalable manner [3]. SSI illusion can be realized in a way as provision of a single process space(SPS). SPS makes a process on any node be able to create child process on the same or different node or communicate with any other process on a remote node, as if they are on a single node. In addition, SPS allows all the processes in the cluster to share an uniform process identification scheme. Thus, a process in the cluster can access all the other processes cluster-wide, using a single namespace. For this purpose, SPS is built with the support of unique cluster-wide pid, signal forwarding.

Several approaches are introduced to provide these functionalities, but they have some problems. In Bproc [4], unique cluster-wide pid does not have scalable performance to fork-intensive programs, and the process migration is not able to move a process with socket. In Opensource SSI Cluster [5], the partitioning of unique pid is not flexible to an application with a variant amount of pid usage. In this paper, we propose a novel design of SPS cluster which addresses the scalability and flexibility problem by using blocked pid allocation. In the blocked pid allocation, a whole pid space is broken up into pieces and the broken pid blocks, not an each pid, are assigned to client nodes. Therefore, the communication overhead for requirement of a new pid is significantly reduced.

The rest of the paper is organized as follows. Section 2 briefly describes some proposed solutions to the prblem of SPS design. Section 3 presents the proposed idea in our design. Section 4 describes the implementation details of our SPS cluster, and Section 5 evaluates the performance of our SPSsystem. We conclude by summarizing our results in Section 6.

2. Related Works

BProc [4, 8] is an approach for provision of single process space based on beowulf cluster, the representative commodity cluster. In Bproc, all the processes running in a cluster are visible on the master node and are controllable via existing UNIX process control mechanisms, since the Bproc adopts the master-slave arcihtecture. With single process space support, the unique pid of a process should not change when it moves, but the Bproc cannot guarantee the fact that a locally assigned pid is globally unique. In order to resolve this problem they make use of pid masquerading, putting a second pid on processes received from the master node or newly forked in slaved node. Therefore, some processes have two pids, local pid and global pid. The operating system runs their works with the local pid as usual and otherwise the global pid is used to identify on the master node if a user wants to control processes on remote nodes. This pid masquerading successfully provides pid globality without much modification in the operating system. However, there are some limitations on Bproc's unique pid support. One or more master nodes manage the entire unique cluster-wide pid space and they allocate an unique pid to slave nodes on demand, instead of that the available pid spaces are given to each node initially. The slave node must contact to server to get a globally unique pid at every fork system

1) This research is supported by National Research Laboratory Grant 4-20.

call. This mechanism requres unwanted communication overhead, and not scalable, especially on fork-intensive programs.

3. Designing Clusters with Single Process Space

In this section, we first describe the new design of SPS cluster and introduce the blocked pid allocation algorithm as the core mechanism to build SPS cluster. Second, we propose a novel idea in unique pid support, what we call blocked pid allocation, and finally, we present a collection algorithm for collecting block with low utilization.

3.1 Design Objectives

We chose design objectives as follows. First, the main objective is to design scalable and flexible single process space cluster. For this purpose, we introduce a new pid allocation scheme, what we call blocked pid allocation. When a node sends a request for a pid, the pool manager replys to the incoming request with a pid block, not inidividual pid as shown in figure 1. The node received this reply can fork some processes within block size without permission of pool manager. Even though the master-slave architecture has been chosen, the blocked pid allocation makes the individual slave nodes can fork some processes without master's admission to a certain extent. Thus, the approach lightens the heavy burden of master node for unique pid allocation, which results in high scalability. Moreover, the pid spaces allocated to individual node are adjusted on the event of new node addition. This offers the ultimate in flexiblity.

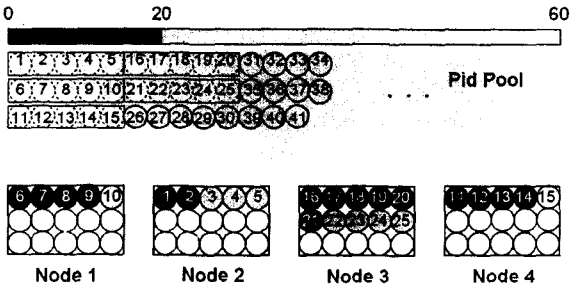

Figure 1 Concept of Blocked PID Allocation

Second, every pids on any node are really unique, not masquerading. The maintenance of mapping local pid and unique pid is complex and it requires additional overhead for adjustment of mapping information at every time a new process is created. The real uniqueness of pid also helps resolving naming conflict in process migration [9]. When a process running with pid 351 moves to another node, the process would need to be assigned a new pid by already existing process with the same pid. If a process which expects the process to have pid 351 calls wait(pid), it will be waiting for the wrong process. With the cluster-widely unique pid, this naming conflict couldnot be problematic, since the pid allocated once on any node is cluster-widely unique. Third, starting a process on remote node should not require logging into the node for convenience. This is similar to Bproc but not exactly the same in that we assume the shared file system, such as NFS. The rsh and process migration is used to implement remote execution for tradional approach and Bproc, respectively. However, we make use of daemon based approach in which the remote execution command are delivered through the daemons and the daemon who received the message spawns a child process and execute the requested excutable. Since, there is no transfer of executable binary with the help of shared file system, we can inituitively think that the overhead of daemon based approach is much less than the rsh or process migration.

3.3 Blocked PID Allocation

In blocked pid allocation, the entire pid space is broken up into pieces. The broken pid blocks are allocated to slave node on demand instead of a single pid. Figure 4 gives the overall picture of blocked pid look-up, with each step of the process labeled. The process begins by blocked pid allocation. The master daemon maintains a pid block allocation table used to record the pid block start number, node number, block usage. Every slave nodes get an initial PID block on system bootup.

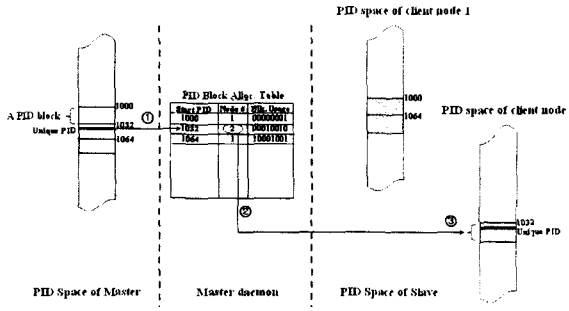

Figure 2 PID look-up on Blocked PID Allocation

4. Implementation

Our SPS Cluster implementation consists of a combination of kernel modification and the user level daemons. The in-kernel portions of SPS Cluster cooporates with user level daemons which follows master-slave architecture. The role of a node in SPS Cluster is only determined by the role of user level daemon. All nodes share the same kernel binary image and there is no need to provide a different configuration for master node. This approach is desirable to minimize the complexity of the kernel.

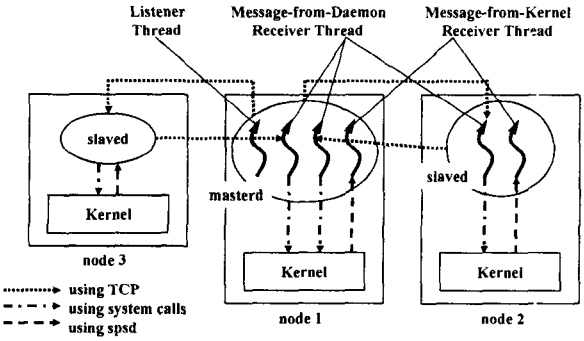

Figure 3 Single Process Space Architecture

Figure 3 shows our SPS architecture in a 3-node cluster, for example. TCP is used in daemon-to-daemon communication, and system calls and Linux character device are used in daemon-to-kernel communication. All nodes have a unique integer ID to distinct each other: master has 1 and slave has an integer larger than 1. When a new slave participates in this SPS, it gets an unique ID.

From the system startup, the masterd notifies the slaved an integer ID and every node gets disjoint pid ranges in which pids are cluster-widely unique. They can freely use theglobally unique pid in the preallocated range. Therefore, this scheme shows scalable performance, since it does not require additional processing, such as communication with master to get permission of use some pid, or checking whether the new pid is globally unique or not, at every process fork time. Unlike the Bproc's approach, the available pid spaces are given to each node initially. This initial allocation of PID space can only be adjusted when a new node participate in SPS or collection of PID block occur.

SPSD : A character device driver for IPC

The kernel sends a message to a daemon with the simple character device to communicate with a user level process. SPSD has a fixed-size

buffer to store messages received from kernel and it allows MKRT to read the message. If a MKRT attempts to read a message even though the buffer is empty, the device driver suspends the MKRT thread until new message is arrived.

Signal Forwarding

kernel must forward signal if there is not a destination process or if it is for multiple destination processes like a process group. The kernel writes a message on spsd to forward it. The message includes a PID of the destination process, a signal number, and a capability of the sending process. Then the MKRT of the slaved is waked up, passes this message tothe masterd, and is suspended again. The MDRT of this slave of the masterd forwards it to the node that a PID block which includes a PID of the destination process is allocated, or to all nodes except this slave in the case of multiple destination processes. Finally the MDRT of a slave which receives this forwarded message calls rkill.

Collection

A PID block starvation problem may occur sometimes in the blocked PID allocation by lowly utilized PID blocks, yet allocated to some other nodes. Collection of low utilized PID block is able to give solution to this problem. Every slave damone receiving the collection message returns lowly utilized PID blocks to the master. A block usage bitmap is exist to represent the block usage information; A bit of the bitmap corresponds to a PID. Before returning lowly utilized PID blocks, kernel sets a flag of process descriptor whose PID is included in the PID block. This flag indicates the fact that the master should be notified on termination of the process for block usage bitmap update.

5. Performance Evaluation

In this section we evaluate the performance of the proposed single process space design. We demonstrate our SPS Cluster's performance by comparing it to Beowulf distributed process space. We used 8 Pentium IV 1.8 GHz machines with 512MB RAM. They are connected by a 100Mbps Fast Ethernet. The implementation of SPS is conducted by modification of Linux kernel 2.4.2. The master-slave daemons runs at user-level on each computation node.

Workload Characteristics

In order to evaluate the system performance, we conducted lmbench [10] microbenchmark.

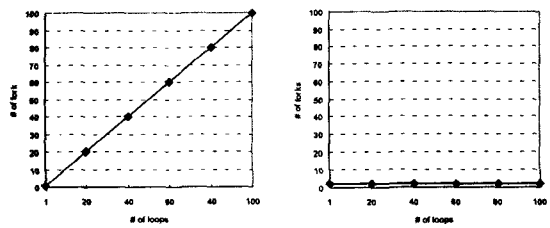

Figure 4 # of processes as function of loops

Figure 4 shows the number of forked processes as a function of the number of loops in the application. The left graph indicate the number of forked processes are linearly increases with the number of loops, while in the right graph the number of newly created processes is always 2 regardless of the number of loops. This means that the former has processes with long life time and the latter benchmark forks new processes which have a short life time.

Figure 5 presents the lmbench microbenchmark execution time as the number of fork system call increases to simulate the case of fork-intensive application. The top graph indicates that our SPS implementation tracks performance of bare linux machine closely, with an observed overhead ranging from 0.2% to 0.6%. The bottom graph shows that overhead of blocked PID allocation increases slightly on the case of long life time process, yet it is still reasonable enough to offer the greatest advantage of blocked PID allocation.

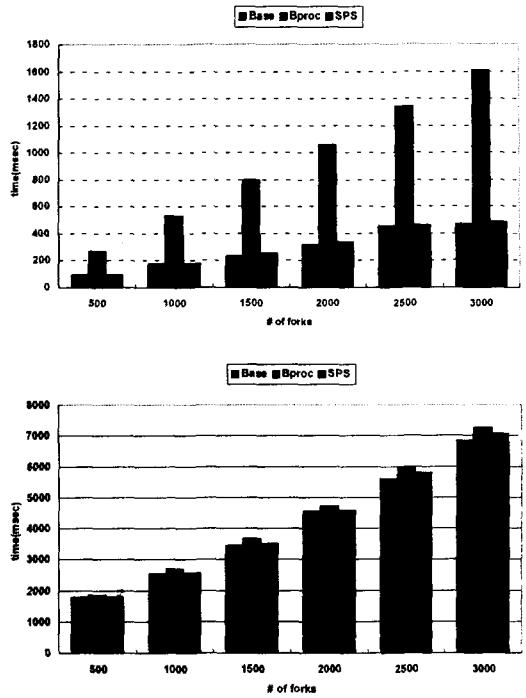

Figure 5 Creation of short life time vs. long life time processes

6. Conclusion

We proposed a novel design of SPS cluster which addresses the scalability and flexibility problem of traditional clusterwidely unique pid implmentation by using blocked pid assignment. Benchmark performance results show that our design of SPS cluster realized both scalability and flexibility.

References

[1] R. Brightwell and S. Plimpton. Scalability and performance of a two large linux clusters, Journal of Parallel and Distributed Computing, 61, 1546-1569, 2001

[2] M. Choi, J. R. Yu, H. J. Kim, S. R. Maeng, Improving Performance of a Dynamic Load Balancing System by Using Number of Effective Tasks, IEEE International Conference on Cluster Computing(CLUSTER), December 01-04, Hong Kong, 2003.

[3] Roy S. C. Ho, K. Hwang, H. Jin, Design and Analysis of Clusters with Single I/O Space, IEEE International Conference on Distributed Computing Systems(ICDCS), April 10-13, Taipei, Taiwan, 2000.

[4] E. Hendriks, BProc: The Beowulf Distributed Process Space, ACM International Conference on Supercomputing(ICS), June 22-26, New York, USA, 2002

[5] B. Walker et al, Single System Image Clusters(SSI) for Linux, http://openssi.org/index.shtml

[6] V. S. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, and E. Nahum. Locality-Aware Request Distribution in Cluster-based Network Servers. In Proceedings of the 8th Conference on Architectural Support for Programming Languages and Operating Systems(ASPLOS), San Jose, CA, October. 1998.

[7] S. Osman, D. Subhraveti, G. Su, and J. Nieh, The Design and Implementation of Zap: A System for Migrating Computing Environments, In Proceeding of USENIX Annual Technical Conference, December 9-11, Boston, MA, 2002.

[8] D. J. Becker, T. Sterling, D. Savarese, J. E. Dorband, U. A. Ranawak, C. V. Packer, Beowulf: A Parallel Workstation for Scientific Computation Proceedings, International Conference on Parallel Processing(ICPP), 1995.

[9] S. Osman, D. Subhraveti, G. Su, and J. Nieh, The Design and Implementation of Zap: A System for Migrating Computing Environments, the 5th Symposium on Operating System Design and Implementation(OSDI), Boston, MA, December 2002.

[10] L. McVoy, C. Staelin, lmbench: Portable Tools for Performance Analysis, USENIX Annual Technical Conference, San Diego, California, January 1996.