Dynamic Load balancing for I/O- and Memory-Intensive workload in Clusters using a Feedback Control Mechanism

Xiao Qin, Hong Jiang, Yifeng Zhu, David R. Swanson

Department of Computer Science and Engineering
University of Nebraska - Lincoln, Lincoln, NE, USA.
Email: {xqin, jiang, yzhu, dswanson}@cse.unl.edu

Abstract. [1] One common assumption of the existing models of load balancing is that the weights of resources and I/O buffer size are statically configured. Though the static configuration of these parameters performs well in a cluster where the workload can be predicted, its performance is poor in dynamic systems where the workload is unknown. In this paper, a new feedback control mechanism is proposed to improve the overall performance of a cluster with I/O-intensive and memory-intensive workload. The mechanism dynamically adjusts the resource weights as well as the I/O buffer size. Results from a trace-driven simulation show that this mechanism is effective in enhancing the performance of a number of existing load-balancing schemes.

1 Introduction

Dynamic load balancing schemes in a cluster can improve system performance by attempting to assign work, at run time, to machines with idle or under-utilized resources. Several distributed load-balancing schemes have been presented in the literature, primarily considering CPU [1], memory [2], or a combination of CPU and memory [3]. Although these load-balancing policies have been very effective in increasing the utilization of resources in distributed systems, they have ignored one type of resource, namely disk I/O. The impact of disk I/O on overall system performance is becoming significant as more and more jobs with high I/O demand are running on clusters. Therefore, we have developed two load balancing algorithms to improve the read and write performance of a parallel file system [4][5].

Very recently, surdeanu et. al [6] developed a load balancing model that considers I/O, CPU, and memory resources simultaneously. In this model, three resource weights were introduced to reflect the significance of resources in a cluster and, as an assumption, the weights of system resources are statically configured.

[1] This paper appears in the proceedings of the 9th International Euro-Par Conference on Parallel Processing (Euro-Par 2003), Klagenfurt, Austria, Aug.26-29, 2003.

To release this assumption that does not hold for clusters with dynamic workloads, this paper proposes a feedback control mechanism to judiciously configure the resource weights in accordance with the dynamic workloads. Moreover, the new mechanism is able to improve the buffer utilization of each node in the cluster when its workload is I/O- and/or memory-intensive.

2 Adaptive Load Balancing Scheme

2.1 Weighted Average Load-balancing Scheme

We consider the issue of a feedback control method in a cluster, $M = \{M_1, ..., M_n\}$, connected by a high-speed network. Since jobs may be delayed because of sharing resources with other jobs or being migrated to remote nodes, the slowdown imposed on a job j is defined as:

$$slowdown(j) = \frac{time_{WALL}(j)}{time_{CPU}(j) + time_{IO}(j)} \tag{1}$$

where $time_{WALL}(j)$ is the total time the job spends running, accessing I/O, waiting, or migrating, and $time_{CPU}(j)$ and $time_{IO}(j)$ are the times spent by j on CPU and I/O, respectively, without any resource sharing.

For a newly arrived job j at a node i, load balancing schemes attempt to ship it to a remote node with the lightest load if node i is heavily loaded, otherwise job j is admitted into node i and executed locally. We propose a weighted average load-balancing scheme, or WAL-FC, where a feedback control mechanism is incorporated. Each job is described by its requirements for CPU, memory, and I/O, which are measured by Seconds, Mbytes, and number of I/O accesses per ms, respectively. For a newly arrived job j at a node i, WAL-FC balances the system load in five steps. First, the load of node i is updated by adding job j's load. Second, a migration is initiated if node i is overloaded. Third, a candidate node k with the lowest load is chosen. To avoid useless migrations, the load discrepancy between node i and k has to be greater than the load induced by job j. If a candidate node is not available, no migration will be carried out. Fourth, WAL-FC determines if job j's migration is able to potentially reduce the job's slowdown. Finally, job j is migrated to the remote node k, and the loads of nodes i and k are updated.

WAL-FC calculates the weighted average load index of each node i, which is defined as the weighted average of CPU and I/O load, thus:

$$load_{WAL} = W_{CPU} \times load_{CPU}(i) + W_{IO} \times load_{IO}(i), \tag{2}$$

where $load_{CPU}(i)$ is CPU load defined as the number of running jobs and $load_{IO}(i)$ is the I/O load defined as the summation of the individual implicit and explicit I/O load contributed by jobs assigned to node i. It is noted that the memory load is expressed by the implicit I/O load imposed by page faults. Let

$l_{page}(i,j)$ and $l_{IO}(i)$ denote the implicit and explicit I/O load of job j assigned to node i, respectively, then, $load_{IO}(i)$ can be defined as:

$$load_{IO}(i) = \sum_{j \in M_i} l_{page}(i,j) + \sum_{j \in M_i} l_{IO}(i). \tag{3}$$

When the node's available memory space is larger than or equal to the memory demand, there is no implicit I/O load imposed on the disk. Conversely, when the memory space of a node is unable to meet the memory requirements, page faults may lead to a high implicit I/O load.

2.2 A Feedback Control Mechanism

To retain high performance, a feedback control mechanism is employed to automatically adjust the weights of resources. The high level view of the architecture for the mechanism is presented in Fig.1, where the architecture comprises a load-balancing scheme, a resources-sharing controller, and a feedback controller. The slowdown of a newly completed job and the history slowdowns are fed back to the feedback controller, which then determines the required control action ΔW_{IO}. ($\Delta W_{IO} > 0$ means the IO-weight needs to be increased, and otherwise the IO-weight should be decreased.) Since the sum of W_{CPU} and W_{IO} is 1, the control action ΔW_{CPU} can be obtained accordingly.

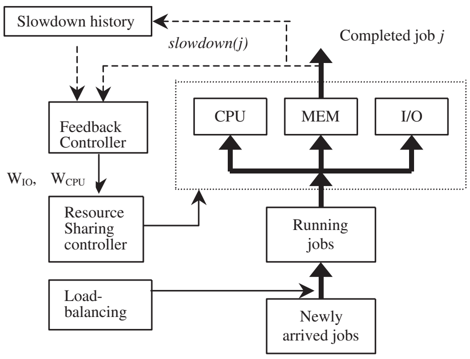

Fig. 1. Architecture of the feedback control mechanism

The feedback controller attempts to manipulate W_{CPU} and W_{IO} in the following three steps. First, the controller calculates the slowdown s_j of the newly completed job j. Second, s_j is stored in the history table, which reflects a specific pattern of the recent slowdowns. The average value of the slowdowns(s_{avg}) in the history table is computed. Finally, the performance is considered being improved if $s_{avg} > s_j$, therefore W_{IO} is increased if it has been increased by the previous control action, otherwise W_{IO} is decreased. Similarly, $s_{avg} < s_j$

means that the performance has been worsened, suggesting that W_{IO} has to be increased if the previous control action has reduced W_{IO}, and vice versa.

Besides configuring the weights, the feedback control mechanism is utilized to dynamically manipulate the buffer size of each node. The main goals are: (1) improve the buffer utilization and the buffer hit rate; and (2) reduce the number of page faults. In Fig.1, the feedback control generates action $\Delta bufSize$ in addition to ΔW_{CPU} and ΔW_{IO}. Specifically, $s_{avg} > s_j$ means the performance is improved by the previous control action, thereby increasing the buffer size if it has been increased by the previous control action, otherwise the buffer size is reduced. Likewise, $s_{avg} < s_j$ indicates that the latest buffer control action leads to a worse performance, implying that the buffer size has to be increased if the previous control action has reduced the buffer size, and vice versa.

3 Experiments and Results

To evaluate the performance of the proposed scheme, We have evaluated the performance of the following load-balancing policies:

(1) CM: the CPU-memory-based policy [3] without using buffer feedback controller(BFC).

(2) IO: the IO-based policy without using the BFC algorithm. The IO policy uses a load index that represents only the I/O load.

(3) WAL: the Weighted Average Load-balancing scheme without BFC [6].

(4) CM-FC: the CPU-memory-based policy in which BFC is incorporated.

(5) IO-FC: the IO-based load-balancing policy to which BFC is applied.

(6) WAL-FC: the Weighted Average Load-balancing scheme with BFC.

In addition, the above schemes are compared with the performance of the following two policies: the non-load-balancing policy without BFC(NLB) and the non-load-balancing policy that employs BFC(NLB-FC).

3.1 Simulation Model

To study dynamic load balancing, Harchol-Balter and Downey [1] implemented a simulator for a distributed system with six nodes. Zhang et. al [3] extended this simulator by incorporating memory recourses. We have modified these simulators, implementing a simple disk model and an I/O buffer model. The traces used in the simulation are modified from [1][3], and it is assumed that the I/O access rate is randomly chosen in accordance with a uniform distribution. The simulated system is configured with parameters listed in Fig.2.

Disk accesses of each job are modeled as a Poisson process. Data sizes of the I/O requests in each job are randomly generated based on a Gamma distribution with the mean size of 250KByte and the standard deviation of 50KByte. Since buffer can be used to reduce the disk I/O access frequency, we approximately model the buffer hit probability for job j running on node i as follows:

$$hit\_rate(i,j) = \begin{cases} \frac{r_j}{r_j+1} \text{ if } d_{buf}(i,j) \geq d_{data}(j), \\ \frac{r_j}{r_j+1} \times \frac{d_{buf}(i,j)}{d_{data}(j)} \text{ otherwise,} \end{cases} \tag{4}$$

Parameters	Value	Parameters	Value
CPU Speed	800 MIPS	Page Fault Service Time	8.1 ms
RAM Size	640 MByte	Seek and Rotation time	8.0 ms
Initial Buffer Size	160MByte	Disk Transfer Rate	40MB/Sec.
Context switch time	0.1 ms	Network Bandwidth	1Gbps

Fig. 2. Data Characteristics

where r_j is the data re-access rate, $d_{buf}(i,j)$ is the buffer size allocated to job j, and $d_{data}(j)$ is the amount of data job j retrieves from or stores to the disk, given a buffer with infinite size. Buffer is a resource shared by multiple jobs in the node, and the buffer size a job can obtain in node i at run time depends on the jobs' I/O access rate and mean data size of I/O accesses.

Figure 3 shows the effects of buffer size on the buffer hit probabilities of the NLB, CM and IO policies. When buffer size is smaller than 150MByte, the buffer hit probability increases almost linearly with the buffer size. The increasing rate of the buffer hit probability drops when the buffer size is greater than 150Mbyte, suggesting that further increasing the buffer size can not significantly improve the buffer hit probability when the buffer size approaches to a level at which a large portion of the I/O data can be accommodated in the buffer.

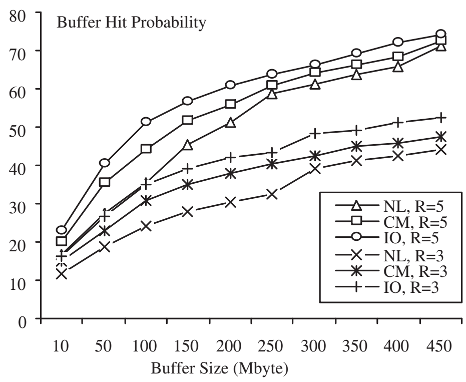

Fig. 3. Buffer Hit Probability. Page-fault rate is 4.0No./ms, I/O access rate is 2.2No./ms, R is data re-access rate.

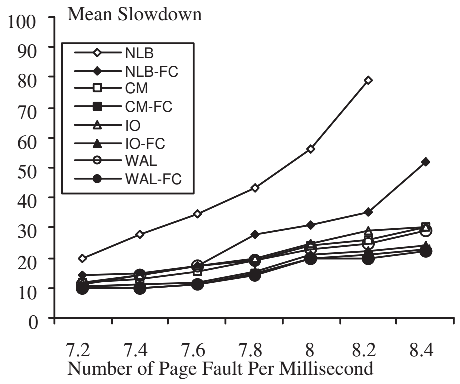

Fig. 4. Mean slowdowns as a function of the page-fault rate. I/O access rate of 1.5No./ms

3.2 Memory and I/O intensive Workload

This section attempts to show an interesting case where the workload is both memory and I/O intensive. The I/O access rate is set to 1.5No./ms, and the page fault rate is from 7.2No./ms to 8.4No./ms.

Not surprisingly, Figure 4 shows that load-balancing schemes achieve significant improvements. Moreover, the performances of CM, IO, and WAL are close to one another, whereas the performance of CM-FC is similar to those of IO-FC and WAL-FC. This is because the trace comprises memory-intensive and I/O-intensive jobs. Hence, while CM and CM-FC take advantage of balancing CPU-memory load, IO and IO-FC can enjoy benefits of balancing I/O load.

A second observation is that, under the memory and I/O intensive workload, CM-based policies achieve higher level of improvements over NLB than IO-based schemes. The reason is that when memory and I/O demands are high, the buffer sizes in a cluster are unlikely to be changed, as there is a memory contention among memory-intensive and I/O-intensive jobs. Thus, the buffer sizes finally converge to a value that minimizes the mean slowdown.

Third, the result shows that the feedback control mechanism can further improve the performance of the load-balancing schemes. For example, WAL-FC further decreases the slowdown of WAL by up to 27.3%, and IO-FC further decreases the slowdown of IO by up to 24.7%. This result suggests that, to sustain a high performance in clusters, compounding a feedback controller with an appropriate load-balancing policy is desirable and strongly recommend.

4 Conclusion

In this paper, we have proposed a feedback control mechanism to dynamically adjust the weights of resources and the buffer sizes in a cluster. The proposed algorithm minimizes the number of page faults for memory-intensive jobs while improving the buffer utilization of I/O-intensive jobs. A trace-driven simulation demonstrates that our approach is effective in enhancing performance of existing dynamic load-balancing algorithms. Future study in this area is two-fold. First, we plan to study the stability of the proposed feedback controller. Second, we will study how quickly the controller converges to the optimal value.

5 Acknowledgments

This work was partially supported by an NSF grant (EPS-0091900), a Nebraska University Foundation grant (26-0511-0019), and a UNL Academic Program Priorities Grant. Work was completed using the Research Computing Facility at University of Nebraska-Lincoln.

This work was partially supported by an NSF grant (EPS-0091900), a Nebraska University Foundation grant (26-0511-0019), and a UNL Academic Program Priorities Grant. Work was completed using the Research Computing Facility at University of Nebraska-Lincoln.

References

1. Harchol-Balter, M., Downey, A.: Exploiting process lifetime distributions for load balancing. ACM Transactions on Computer Systems **15** (1997) 253–285

2. Acharva, A., Setia, S.: Availability and utility of idle memory in workstation clusters. In: Proceedings of the ACM SIGMETRICS Conf. on Measuring and Modeling of Computer Systems. (1999)
3. Zhang, X., Qu, Y., Xiao, L.: Improving distributed workload performance by sharing both cpu and memory resources. In: Proceedings of the 20th Int'l Conf. on Distributed Computing Systems. (2000)
4. Zhu, Y., Jiang, H., Qin, X., Feng, D., Swanson, D.: Improved read performance in a cost-effective, fault-tolerant parallel virtual file system (ceft-pvfs). In: Proc. of the 3rd IEEE/ACM Intl. Symp. on Cluster Computing and the Grid. (2003) 730–735
5. Zhu, Y., Jiang, H., Qin, X., Feng, D., Swanson, D.: Scheduling for improved write performance in a cost-effective, fault-tolerant parallel virtual file system (ceft-pvfs). In: the Fourth LCI International Conference on Linux Clusters. (2003)
6. Surdeanu, M., Modovan, D., Harabagiu, S.: Performance analysis of a distributed question/answering system. IEEE Trans. on Parallel and Distributed Systems **13** (2002) 579–596