A Novel Approach for Effective Dynamic Load Balancing by using Tendensive Weight

K. Shahu Chatrapati, K. Purna Chand, Y. Ranjith Kumar, Dr. A. Vinaya Babu

Abstract:
Clusters have emerged as a primary and cost-effective infrastructure for parallel applications, including communication-intensive applications that transfer a large amount of data among nodes of a cluster via interconnection networks. Efficient resource usage is a key to achieving better performance in cluster systems. Previously, most research in this area has focused on balancing the load of each node to use the resources of an entire system more effectively. Memory utilization has a key role in the implementation of a high performance cluster. However, determining the load without considering the memory utilization will degrade the cluster performance. In this paper, a load metric has been proposed, termed as Tendensive Weight, in order to balance the cluster dynamically. This metric considers both CPU and IO utilization including memory exploitation. Simulation results show that the system incurs 10% shorter execution time than the conventional approach using number of effective tasks.

Key Words:
HPC, Load Balance, CPU utilization, Memory exploitation, Tendensive Weight.

INTRODUCTION:
Scheduling and load balancing are two key techniques used to improve the performance of clusters for scientific applications by fully utilizing machines with idle or under-utilized resources. A number of distributed load-balconing schemes for clusters have been developed, primarily considering a variety of resources, including CPU [10], memory [1], disk I/O [6], or a combination of CPU and memory [3]. These approaches have been proven effective in increasing utilization of resources in clusters, assuming that networks connecting nodes are not potential bottlenecks in clusters. However, a cluster system has the tendency to concentrate the system load on to certain nodes, resulting in coexistence of overloaded nodes and idle resources. Therefore, the development of a load balancing system for utilizing computing resources of lightly loaded nodes is crucial to resolving the problem of load imbalance in the cluster system.

A large number of scientific applications have been implemented for executions on clusters and more are underway. Many scientific applications are inherently computation and communication intensive [7]. Examples of such applications include 3-D perspective rendering, molecular dynamics simulation, and quantum chemical reaction dynamics simulations and 2-D fluid flow using the vortex method, to name just a few [7]. The above bottleneck becomes more severe if the resources of a cluster are time-/space-shared among multiple scientific applications and memory load is not evenly distributed among the cluster nodes. Furthermore, the performance gap between CPU and memory continues to widen at a faster pace, which raises a need of increasing the utilization of memory on clusters using various techniques.

The load balancing of an application has a direct impact on the speedup to be achieved as well as in the performance of the parallel system [8][11]. Usually, when working with heterogenous clusters, the different calculation powers of the intervening machines are a factor that can be computerized to analyze the distribution of the work to be done. For the type of known work problems (e.g. matrix multiplication) a “predictive” static load balancing considering the calculation power of the multi cluster processors can be obtained; however, many real problems have a variable or dynamic workload depending on the data [2][3][10]. In these cases, it is necessary to adjust data or processes allocation dynamically while the application is being executed. Note that any “predictive” load balancing formula should compute not only the calculation power, but also other factors proper of the architecture, such as the size and access time at the different levels of memory of each processor [1][2].

The traditional approach only considers the calculation power of each processor has been taken into account. Besides, in a multi-cluster scheme in which applications are resolved with the Master-Slave paradigm, any dynamic balancing solution used, implies a performance overhead that will be affected by the complexity of the initial job placement scheme among the nodes of the different clusters, as mentioned before.

In this paper, a load metric Tendensive Weight is proposed which contains information about both the system load and resource utilization, that includes CPU and IO utilization in addition with memory access rate(MAR). And a load balancing algorithm is developed, in which the load of each node is considered using the metric termed as...
Tendensive Weight. For experimental results, this algorithm is compared with the traditional load balance algorithm which considered only CPU utilization with no of effective tasks as load metric, our algorithm given away significant improvement in system performance.

The remainder of the paper is organized as follows. In the next section, we provide the related work regarding our load metric; Section 3 describes the concept of the dynamic load balancing with Tendensive Weight. Section 4 reports the advantages of the proposed system and provides a comparison with the initial job placement with the effective tasks. Finally, Section 5 describes an algorithm to calculate the Tendensive Weight and in the next section we evaluate max performance improvement using Tendensive Weight and finally conclude with a summary.

RELATED WORK:

In general, load-balancing techniques fall into two camps: centralized and distributed load balancing. Centralized schemes require a head node that is responsible for handling load distribution. As cluster size scales up, the head node quickly becomes a bottleneck. To solve this scalability problem, distributed dynamic load balancing can be used to delegate workload of load balancing to multiple nodes in a cluster. In addition, a centralized scheme has a potential problem of poor reliability because permanent failures of the central load balancer can render the entire load balancing mechanism dysfunctional. Therefore, the focus of this paper is on designing a decentralized communication-aware load balancer for time-/space-shared clusters.

Paul werstien and et. al.[1] proposed a dynamic load balancing algorithm which considers CPU length, CPU and memory utilization and network traffic as load metric. In this they compared the results with algorithm that is using only queue length as metric.

Min Choi, Jung-Lok Yu and et. [2] Al. projected a Dynamic Load Balancing System by Using Number of Effective Tasks propose a novel load metric termed number of effective tasks in order to resolve the problem arising from inaccurate predictions. Thus, the initial job placement system can work without knowing job resource usage in priori.

Xiao Qin and Hong Jiang [14] developed a Load balancing strategy for Communication–Intensive applications to improve the band-width across network of cluster. This scheme can make use of an application model to quickly and accurately determine the load induced by a variety of parallel applications.

A communication-sensitive load balancer has been proposed by Cruz and Park [6]. The balancer uses run-time communication pattern between processes to balance load. Orduña et al. have proposed a criterion to measure the quality of network resource allocation to parallel applications [14]. Based upon the study of all above thesis we proposed a load metric “Tendensive Weight”, this metric considers both CPU and IO utilization including memory access rate. And we developed an algorithm for calculation of the Tendensive Weight of each task to be distributed to the node.

DYNAMIC LOAD BALANCING SYSTEM WITH TENDENSIVE WEIGHT:

“Tendensive Weight” refers to the number of tasks that actually affect system performance. Observing that overlapping the execution of CPU bound and I/O bound jobs with the efficient memory utilization i.e. efficient memory access rate yields improvement in resource utilization, we focused on the aspect that these two kinds of jobs do not affect each other’s execution time. The resource utilization is closely related to the system performance. In other words, the Tendensive Weight can be thought of as a kind of load metric that considers the overlapping effect.

Figure 1 illustrates the local memory reference between the number of CPU-Bound and IO-Bound tasks. There are five running jobs: three for CPU bound jobs and two for I/O bound jobs. In this case, the system load is shown as the number of tasks in the systems, which are 5. However, according to the number of tasks, the system load is 3+ by counting both CPU bound and I/O bound jobs as one. There is no significant difference in execution time between when 1 CPU bound job or 1 I/O bound job runs alone and when they run together simultaneously. However, the resource utilization of the latter has been improved. Because the load metric Tendensive Weight considers both the number of running jobs and the resources i.e. CPU, IO and memory access rate, it better reflects the system load than the traditional method of only considering the number of tasks executing at a given time distributed over nodes on a heterogeneous clusters.

Advantage of the Tendensive Weight:

Information about the resource requirement of a job that is to be assigned is necessary for the initial job placement system to maximize the resource utilization of each node and the entire system performance too. This is because the most appropriate node to execute the task is determined by the kind of resources the job requires during the execution.
However, if we espouse the Tendensive Weight as a load metric, we can avoid inappropriate placement even if we lack such prior knowledge in the initial stage. In Figure 2, a newly arrived job is placed while three jobs are running on each node. Here the coming job may be a CPU bound or IO bound task.

![Figure 2: Initial Job Placement referencing local Memory](image)

Jobs that require the same resources are placed in a single node. The second block shows the best case job placement, wherein the number of jobs that require a type of resource is the same in a node. And the third block represents the possible outcomes when we use initial job placement utilizing an estimation based on historical data. As noted in Section 2, all the feasible outcomes can occur due to erroneousness in the estimation. Final block represents the possible cases of initial job placement system using the Tendensive Weight. Clearly, if we have the information about the resource constraint of the newly arrived job, we can always achieve the best job placement. However, since it is almost impossible to achieve such information in realism, the system employing the number of effective tasks may cause an imbalance in the resource usage as in the case of K. However, our system Tendensive Weight based approach can avoid the significant performance degradation. And attains max performance improvement over traditional one. This is the greatest advantage of using the Tendensive Weight.

**Calculation of the Tendensive Weight:**

At every booting point of a system the Tendensive Weight is always calculated at every point of time when time quantum ends. Always the time quantum should be generated by the CPU cycles. In a single time quantum, every process has a specified time quantum that is computed at the beginning of CPU utilization. This is the maximum CPU time that the process can use during it execution. The time quantum ends when all run able processes have used their entire quantum. A new time quantum then starts, and all processes obtain a new time slice. For the use to calculate the Tendensive Weight, we first initiate the concept of the I/O to CPU ratio. The I/O to CPU ratio is a ratio of the average CPU time consumed by an I/O bound job to the average CPU time consumed by a CPU bound job and the average memory access rate of each task assigned to a node in a cluster.

\[
\text{IO to CPU Ratio} = \frac{AVG(t_{cpu}(io_i))}{AVG(t_{cpu}(cpu_i)) + AVG(t_{mem}(io_i))}
\]

For the remainder of this paper, we use the I/O to CPU ratio as defined here. If the I/O to CPU ratio is large, the I/O bound job consumes a large amount of CPU resource. If not, the I/O bound job only uses a small amount of CPU resource. Overlapping the I/O bound job and the CPU bound job rarely improves the system performance for a small I/O to CPU ratio. Since both types of jobs require the CPU resource simultaneously, the contention results in performance degradation. In this case, we do not have to calculate the number of effective tasks. Instead, we just return the number of tasks as the number of effective tasks. As a result, when the I/O to CPU ratio is relatively large, the number of effective tasks has the same value as the number of tasks.

If IO to CPU Ratio >= 0.5 then

\[
\text{Tendensive Weight (TW)} = \text{TW (IO)} + \text{TW (CPU)}
\]

On the other hand, when the I/O to CPU ratio is small, the overlapping effect is quite high. The area that has a larger number of processes of identical type, whether it is CPU bound or I/O bound, becomes more influential to the system performance. In other words, when CPU bound jobs outnumber I/O bound jobs, the system load is mainly determined by the CPU time, which is consumed by the executing jobs in an epoch. When the number of CPU bound jobs is larger than the number of I/O bound jobs, the number of effective tasks is derived by dividing the sum of all consumed CPU times of the entire tasks by the average consumed CPU time and the average memory access time consumed by all the jobs. Finally the I/O to CPU ratio is large then the Tendensive Weight calculated as the sum of Tendensive Weight of IO-bound tasks and the Tendensive Weight of CPU-bound tasks.

\[
T \ W (CPU) = \frac{\sum t_{cpu}(job_i)}{AVG(t_{cpu}(cpu_i)) + AVG(t_{mem}(cpu_i))}
\]

\[
= \frac{\sum t_{cpu}(cpu_i) + \sum t_{cpu}(io_i)}{AVG(t_{cpu}(cpu_i)) + AVG(t_{mem}(cpu_i))}
\]

\[
= n(cpu) + \frac{\sum t_{cpu}(io_i)}{AVG(t_{cpu}(cpu_i)) + AVG(t_{mem}(cpu_i))}
\]
\[ T_W(IO) = \frac{\sum t_{io}(io)}{AVG(t_{io}(io)) + AVG(t_{mem}(io))} \]

\[ = \frac{\sum t_{io}(io) + \sum t_{io}(cpu)}{AVG(t_{io}(io)) + AVG(t_{mem}(io))} \]

\[ = n(io) + \frac{\sum t_{cpu}(cpu)}{AVG(t_{cpu}(cpu)) + AVG(t_{mem}(cpu))} \]

Finally, Tendensive Weight is calculated as no of I/O bound tasks, number of CPU bound tasks and the ratio of average CPU time \( t_{cpu} \) consumed by the IO bound tasks to the average CPU time \( t_{cpu} \) consumed by the CPU bound tasks in addition with average memory access time \( t_{mem} \) of a CPU bound task.

**SPECIFICATIONS:**

- \( N(io) \) = No of I/O tasks;
- \( N(CPU) \) = No of CPU tasks;
- \( t_{cpu} \) = Process time or CPU time;
- \( t_{io} \) = IO time to assign a task;
- \( t_{mem} \) = Time taken for memory access

**IMPLEMENTATION:**

The implementation of the load metric 'Tendensive Weight' starts from this point. The algorithm for calculation of the Tendensive Weight is as follows. The algorithm captures the task to be executed on appropriate node, if the task to be executed is a IO-bound task then the counter incremented and the TW (IO) is calculated or otherwise if the task is CPU-bound task then the counter of CPU-bound tasks is incremented and TW(CPU) is calculated. Finally, by considering the Tendensive Weight(TW) the task is distributes to the proper node in a cluster.

**Algorithm:**

Read_lock (&task_list_lock)

For_each_task(p)

if ((p->state == TASK_RUNNING || p->state == TASK_UNINTERRUPTIBLE))

\[ Pr = Tendensive Weight(IO); \]

if ((Tendensive Weight(IO) != 0 && Tendensive Weight(CPU) != 0))

\[ Compute I/O to CPU ratio; \]

if (I/O to CPU ratio is < 0.5)

if (Tendensive Weight(IO) > Tendensive Weight(CPU))

\[ Pr = Tendensive Weight(IO); \]

else if (Tendensive Weight(IO) < Tendensive Weight(CPU))

\[ Pr += 1; \]

Compute CPU factor;

Compute Memory Access Rate i.e. MAR;

if (the job is I/O bound)

\[ Tendensive Weight (IO) = I/O bound tasks++; \]

\[ total CPU quantum and MAR used by I/O bound tasks += p->quantum used; \]

else if (the job is CPU bound)

\[ Tendensive Weight (CPU) = CPU bound tasks++; \]

\[ total CPU quantum and MAR used by CPU bound tasks += p->quantum used; \]

\} Read_unlock (&tasklist_lock);

return Pr;

**Performance Evaluation:**

For the simulation results, we evaluate the performance of the proposed dynamic load balancing system with the Tendensive Weight. We used a Pentium Dual-Core 3.2 GHz machine with 2 GB RAM as a global job scheduler with 8 computation nodes. Each node is a Pentium IV 2.8 GHz machine with 512MB RAM. They are connected by a 100Mbps speed, twisted pair Ethernet i.e. 10Base-T/100Base-TX Dual speed Hub. Our system attains a maximum improvement in performance over the traditional method considering number of tasks as a load metric.
Maximum Performance Improvement:

The simulation is performed twice by two systems: one using estimation based on the number of effective tasks and another with Tendensive Weight. The performance metric is the execution time and the total execution time. The former system places a new job to the node that has the least value of the number of effective tasks. The latter system estimates the required resource of a job to be assigned. It assumes that the new job which has just arrived is identical to the job previously executed, if the job name and the parameters match. If the newly arrived job is an unprecedented execution, the system cannot correctly estimate the required resource of the job. Even if the new job matches the previous job, it is possible that the job now requires different resource from the last execution. Therefore, the estimation based on the number of effective tasks may be incorrect and the inaccurate estimation can result in performance degradation.

However, our system using Tendensive Weight can avoid performance degradation resulting from misplacement of a job even when the job has not been previously executed. Figure 5 shows a comparison of execution times in the traditional based estimation using the number of effective tasks-based approach to the our proposed Tendensive Weight based approach. This is because the Tendensive Weight is meaningful only when the number of currently running jobs on a node is more than 2. Otherwise, the Tendensive Weight has the same value as the number of tasks. In this case, the job is placed on the most appropriate node in our proposed system, while the job is placed to an inappropriate node in the Traditional based approach because of wrong estimation. On the other hand, the traditional based approach correctly estimated the required resource for tests 2 and 3 i.e. despite this failure; it has placed the jobs to relatively adequate nodes, which enables the system to avoid a high increase of execution time by wrong estimation. As a result by considering the Test1 a performance improvement of up to 10% for the ideal case is observed in the system with Tendensive Weight.

Performance improvement by considering the CPU utilization under a Test Case:

Construction of the exact same execution environment as that on the machine from which the trace data is collected is impossible. Instead, we created a Test Case and the CPU utilization of each task. This is able to adjust CPU utilization and run for any specific period of time. Our Load balancing algorithm calculates the each task’s CPU time, IO time and memory access rate (MAR) with our proposed effective load metric ‘Tendensive Weight’. To reflect the various execution environments, we conducted the same simulation for four different Tests. Each and every test is conducted on different homogeneous operating environments and the simulation results are taken. Here the calculated values at different test are shown in Table1.

<table>
<thead>
<tr>
<th>Test Cases</th>
<th>CPU Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test 1</td>
<td></td>
</tr>
<tr>
<td>CPU Task</td>
<td>0.998%</td>
</tr>
<tr>
<td>IO Task</td>
<td>0.195%</td>
</tr>
<tr>
<td>MAR</td>
<td>0.61µs</td>
</tr>
<tr>
<td>Test 2</td>
<td></td>
</tr>
<tr>
<td>CPU Task</td>
<td>0.805%</td>
</tr>
<tr>
<td>IO Task</td>
<td>0.198%</td>
</tr>
<tr>
<td>MAR</td>
<td>0.45µs</td>
</tr>
<tr>
<td>Test 3</td>
<td></td>
</tr>
<tr>
<td>CPU Task</td>
<td>0.999%</td>
</tr>
<tr>
<td>IO Task</td>
<td>0.234%</td>
</tr>
<tr>
<td>MAR</td>
<td>0.52 µs</td>
</tr>
<tr>
<td>Test 4</td>
<td></td>
</tr>
<tr>
<td>CPU Task</td>
<td>0.899%</td>
</tr>
<tr>
<td>IO Task</td>
<td>0.296%</td>
</tr>
<tr>
<td>MAR</td>
<td>0.42 µs</td>
</tr>
</tbody>
</table>

Table1. Four Tests of workload having different degree of CPU utilization

Process that consumes between 40% and 60% of the CPU cannot be classified as a CPU bound or I/O bound job. Test1 displays the maximum performance improvement, as described in previous section.

And the performance improvement of proposed scheme over the traditional task based approach can be calculated and attains a maximum performance improvement upon 10% over a traditional task based approach with no of tasks. The performance improvement of our Tendensive Weight based approach upon different Tests is shown in Table 2.

<table>
<thead>
<tr>
<th>Test</th>
<th>Task Based</th>
<th>Tendensive Weight</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test 1</td>
<td>85.34%</td>
<td>96.24%</td>
<td>10.9%</td>
</tr>
<tr>
<td>Test 2</td>
<td>74.67%</td>
<td>77.29%</td>
<td>2.6%</td>
</tr>
<tr>
<td>Test 3</td>
<td>92.85%</td>
<td>99.59%</td>
<td>6.74%</td>
</tr>
<tr>
<td>Test 4</td>
<td>90.62%</td>
<td>98.74%</td>
<td>8.12%</td>
</tr>
</tbody>
</table>

Table2. Total execution time and performance improvement
Figure 4 is histogram for this data. The workload Test 4 consists of less CPU bound jobs and less I/O bound jobs, while the workload Test 1 consists of more CPU bound tasks and more I/O bound tasks. As a result, a greater amount of resource bound tasks results in greater performance improvement.

![Efficient Performance by Tendensive Weight](image)

**Figure 4: Schema for Performance improvement by Tendensive Weight.**

**Conclusions:**

We propose a novel load metric called the Tendensive Weight. The Tendensive Weight is the number of tasks that actually affect system performance. It contains information about both system load and resource utilization simultaneously. This metric is especially useful for use in a dynamic load balancing system. A system using the Tendensive Weight can work without prior knowledge about job resource requirement. As such, the system no longer depends on the estimation. Therefore, it does not suffer performance degradation due to wrong estimation. The system eliminates cases where the execution time is increased up to 2-3 times as a result of assigning the job to an inappropriate node. Simulation results show that the dynamic load balancing system with Tendensive Weight provides performance improvement of up to 10% compared to the estimation based approach.

**References:**


K. Shahu Chatrapati Pursuing his PhD (Computer Science and Engineering) from Acharya Nagarjuna University, Guntur, Andhra Pradesh, India. He has received the B.E degree from Bapatla Engineering College in 1999 and M.Tech degree from College of Engineering, Andhra University in Computer Science and Engineering, in 2002. He is in teaching and research since 2002. He has published 2 papers in International conferences/journals. His research is focused on Load Balancing in High Performance Clusters. He is working as an Assistant Professor in JNT University, Kakinada, INDIA.

K. Purna Chand Pursuing his M.Tech in (Computer Science & Engineering) degree from JNT University, Kakinada, India. He has received the B.Tech degree in Computer Science and Engg from Narasaraopeta Engineering College Narasaraopet, Andhra Pradesh, India. His research is focused on Load Balancing in High Performance Clusters.

Y. Ranjith Kumar received M.Tech degree in Information Technology from College of Engineering, Andhra University in 2002. He is in teaching from past 10 years. He has published 2 papers in National conferences/journals. His areas of interest are Computer Networks, Network Security, Mobile Networks & its applications, and he is life member of ISTE. He is working as a Professor in CSE, DMSSVH College of Engineering, Machilipatnam, INDIA.

Prof. Dr. A. Vinaya Babu received PhD in (computer Science) from JNT University Hyderabad, India. He has received BE and ME from Osmania University in Electronics from JNT University in Computer Science and Engineering. He is a professor in CSE and Director of School of Continuing and Distance Education in JNT University, Hyderabad, India.