Enhanced Linux Load Balancer for Multi-core Processors on the basis of Average Weight of Tasks

Anurag Sharma
Rajiv Gandhi Proudyogiki Vishwavidyalaya
School of Information Technology, India
462033
aanurag4321@gmail.com

Jitendra Agrawal
Rajiv Gandhi Proudyogiki Vishwavidyalaya
School of Information Technology, India
462033
jitendra@rgtu.net

***Abstract*— Due to the rapid growth of computers and high performance demand by users, single core processors are being replaced by multicore processors. Multicore processors are gaining popularity day by day as they contain two or more cores, placed on the single chip, enabling execution of multiple tasks present in the system. The distribution of these tasks across multiple cores such that the cores are equally utilized, is referred to as load balancing. In order to efficiently utilize the processing cores and to improve system performance, it is important to optimize the existing load balancing algorithms used by operating systems. Existing load balancer of Linux assigns an equal number of tasks to each core. Despite this, load imbalance problem remains because the CPU usage of tasks may vary and size and type of processes may be different and hence workload gets imbalanced in very less time, requiring the load balancing to be done frequently. Therefore, the Linux load balancer performs task migration dynamically, from one core to another core in order to maintain the load balance across the cores, however, this task migration results in an overhead. Thus to minimize the overhead of task migration, the assignment of the task should be based on the type and size of the processes. To improve the system performance, therefore, a new load balancing approach is proposed. In this approach, an Average Load Balancer (ALB) balances the load among cores on the basis of type and size of the process i.e. weight of the tasks. Due to Average Load Balancer the frequency of invoking load balancer gets reduced and thus the system performance is improved. The various tests are performed for single and multiple workload by Phoronix-test-suite tool. The results show that the ALB improves the performance on an average by approx 8% in terms of MIPS and approx 5% in terms of execution time as compared to the existing load balancer.**

Keywords: *Multi-core Processors, Load Balancing, Average weight*

I. INTRODUCTION

As personal computers have become more prevalent and more applications are being designed for them, the end user has seen the need for a faster and more capable systems to keep up. In single core architecture, speedup has been achieved by increasing clock speeds but there is a limitation to increase clock frequency. Another way to achieve speedup is to add multiple processing cores to a single chip called multicore architecture.

A multicore processor is a single computing component with two or more independent actual processing units called cores, which are the units that execute program instructions. The multiple cores can run multiple instructions at the same time, increasing overall speed of program execution. The cores are typically integrated onto a single integrated circuit die known as chip multiprocessor or CMP.

The use of multicore architecture has rapidly increased to develop processors as it improves speed by adding multiple processing cores to the single chip. However, adding multiple cores to the single chip processors have many challenges related to memory, cache coherence, power consumption, load imbalance among multiple cores etc.

Among these challenges the load balancing between the cores is one of the big challenges. To equally utilize each core, the workload among cores must be assigned equally. If the load becomes imbalanced across the cores, the tasks will be migrated from overloaded core to the least loaded core. While migrating the task from one core to another core, there is need to find out load imbalance. At the time of finding imbalance we used the Average Weight condition for task migration, which helps to balance the load.

The multicore processor has very big challenge of improving the performance. The performance of the multicore system can be improved by increasing MIPS or decreasing the execution time of the processes. Multicore processor has several cores which are responsible to execute the task. So, all the cores must be fully utilized, i.e. the load on the different cores must be balanced.

The task scheduler has responsibility to allocate the task on each core whereas load balancer has responsibility to balance the load on each core. In the Linux kernel 4.4.1 load balancer is called when the CPU becomes idle, or if fork() or exec() is executed, or if task wakes up. Load balancer periodically checks whether the load is balanced or not. If the load is imbalanced then the tasks will be migrated from the busiest core to the core which has the less load compared to the busiest core. Hence, we need to perform task migration. However, this task migration results in an overhead, and we should perform load balancing in such a way that the task migration overhead is reduced.

The existing Linux load balancer performs the load balancing, but there is possibility to improve the load balancing performance. Since at runtime the task priority may change due to the type and the size of the process, the tasks should be distributed across the cores on the basis of their sizes and type. But in existing Linux load balancing tasks are assigned to each core equally in number. The type of the process may be either CPU bound or IO bound, which is decided based on the time slice for which it runs on the CPU. A process is considered a CPU bound process if the process consumes full default time slice allocated at first time for execution. Every such CPU bound process is penalized with 'nice' value. The nice value of the process is inversely proportional to the CPU usage of the process. If nice value is increased by some value then the

usage of CPU percentage will decrease. In the existing approach the tasks are not fairly distributed

on different cores, because of different CPU usages. Due to this reason, workload gets imbalanced in lesser time, requiring the load balancing to be done frequently. This in turn degrades the overall system performance. The aim of the proposed load balancer is to perform load balancing on the basis of the average weight of the tasks.

II. Related Work

In [8] Bautista proposed a simple power-aware scheduling algorithm for multicore systems. This scheduling algorithm moves the extra workload from overloaded cores to the less loaded cores. A complete task is moved from overloaded system to less loaded cores. This algorithm reduces the energy consumption while increasing or decreasing the frequency of the cores. Simple power aware technique is never able to maintain the workload equal on cores, so all cores are not equally utilized. This scheme does not split the task so it moves the complete task to other core. Hence, it requires the task migration which in turn incurs overhead.

In [9] Raj Kumar introduced the technique of highest priority task splitting. Task is divided into two portions t1 and t2. of each splitting task is assigned to next processing core and t2 is assigned to the last core. Every time a task is allocated to a processing core, a schedulability test ensures that the tasks allotted to a core are schedulable with deadline monotonic. The existing problem in this approach is that all cores are not equally loaded. Hence it requires task migration to equalize the load.

In [10] N. Min Allah proposed a scheduling algorithm which finds the least loaded core and then the lightest task from the heavy core is shifted to the least loaded core to maintain the uniform workload on system. The problem in this approach is that there is no task splitting and all cores are not equally utilized. Therefore to shift the task we need to perform task migration.

Kato [12] presented a partitioned scheduling scheme for the multiprocessors. This technique assigns the tasks to specific processors an such a way that processor one is filled with tasks 100 percent utilized and remaining processor are filled according to some specific value. A task can be split in two subtasks and these two subtasks are assigned to different processors. Split tasks are executed in any order. All processors are not equally utilized because processor 1 is 100 percent utilized and the remaining processors are utilized to some specific value.

Suchi Johari and Arvind Kumar [1] proposed a method for load balancing named as average method. The method is best solution for load balancing. The average number of tasks is taken as the threshold value. Threshold value refers to average number of tasks per core. It balances the load on cores. This technique provides guarantee of 99% perfect load balancing during task migration but still it has a limitation. If tasks are integral multiple number of cores then it works perfectly otherwise it is required to search the core which has some empty space to accept the task [1].

$$Thresholdvalue = \frac{Totalnumberof\ taskspresent}{Numberof\ cores}$$

$$Averagenumberof\ taskpercore =< ThresholdValue$$

The Average method gives the best solution for the load balancing, because this method assigns the equal number of the tasks to each core. But, practically every tasks has the different CPU usage. So at run time the load on each core will be changed. Therefore load may become imbalanced. Hence in proposed Linux Load Balancer the average number of tasks on each core is calculated based on their CPU usage.

In this chapter we discussed work done by different researcher for load balancing. All the existing load balancing techniques have some limitations. These techniques attempt to maintain equal workload on all cores but their performance is not optimum as each technique requires frequent task migrations resulting into lots of overheads. So, to overcome the task migration overheads, we have optimized the existing Linux load balancing technique by reducing the number of task migrations, using the concept of average weight of the tasks.

III. Proposed Approach

This algorithm is an efficient solution for solving load imbalancing problem on different cores. It balances the load on different cores and keeps minimum difference of load across the cores. In this algorithm the Average weight value is calculated. This Average Weight value is based on the CPU Usage. The CPU usage of any process is calculated on the basis of the processes weight. The Linux Completely Fair Scheduler (CFS) calculates a weight based on the nice value. The weight is calculated as $1024/(1.25^{nice\_value})$. As the nice value decreases the weight increases exponentially. The implementation of the CFS is in kernel/sched/fair.c. Nice parameter is used to calculate the average weight value. Average Weight value will help in load balancing.

The Average weight is calculated as sum of weight of the task present in the run queue divided by the number of processing units. The total weight sum of maximum number of task on i^th core should be less than or equal to the Average weight.

This algorithm will help to balance the workload among the cores. If processes weight sum is multiple of number of cores then it works perfectly otherwise an extra effort is required for searching the core which has empty space to migrate the new task.

- Algorithm for Load Balancing

- Find the number of cores present in the multicore processor i.e. cpu count
- Calculate Average Weight = sum of weight of the tasks present in the run queue / cpu count;
- Algorithm:
  - Assign the tasks to each core until the sum of weight of all the assigned tasks is less then the average weight
  - Repeat above process until all the core are not accessed.

A. Implementation Details

The linux kernel development required the source code which is available on www.kernel.org and the linux cross reference site is also available to see the code. The major code of load balancing and scheduling is in the following files /source/kernel/sched/*.c

1) **/source/kernel/sched/rt.c:** For realtime tasks and the code related to these is available in this file.
2) **/source/kernel/sched core.c :** Number of logical CPU details in group and performing load balancing and other related operation.
3) **/source/kernel/sched fair.c :** Code related to the scheduling is written in this file and other data member and member functions are defined here.
4) **/source/kernel/sched sched.h :** This header file is used at various places in source code.
5) **/ include/linux/interrupt.h/:** This is used to initialize and handle software interrupt.
6) **/source/include/linux/jiffies.h:** This file contains the different methods and jiffies value for major time interval.

Multicore processor has a separate run queue for each core. Each core selects processes only from its own runqueue to run. The main data structure used to access the per- CPU runqueue struct rq is the data structure used in Linux that contains all the information about a specific runqueue including no. of running tasks. Structure rq is defined in ¡include/linux/sched.h¿.

IV. Testing and Results

This chapter explains all the experimental details. The first section explains the experimentation details, that is, how to setup the system and experiment method for single workload as well as multiple workload . Second section presents the results and compares the same with existing Linux load balancer. Third section gives the interpretation of the results followed by the outcome of the experiments.

A. Steps Used for Experimentation

To imbalance the load on the particular core and to check the outcome, we need to follow these steps:

- **STEP 1**: First install the Phoronix-test-suits
- **STEP 2**: Install various test cases, like 7-Zip compression, flace audio encoding for CPU intensive tasks, IO intensive tasks, and Memory Intensive tasks etc.
- **STEP 3** : Run any CPU Intensive, or IO intensive task for example 7-zip Compression which is CPU intensive task.
- **STEP 4** : Find, what is the Process ID of 7-zip Compression process or any other process with the help of 'top' command interface.
- **STEP 6** : After finding the process id, assign that task to the particular core with the help of process id. Then the current affinity of the task will change. It means that the task will execute on selected core.
- **STEP 7**: Analyze the output generated by the bench mark which we have used for the corresponding task. The task output varies corresponding to the task type, it may be either in MIPS or in Seconds.

B. Single Work Load and Multiple Work Load

The single workload means providing the load through single process, it may be either CPU bound or IO bound, for example 7zip, Flac-audio, Gzip etc. These processes are available in phoronix test suite. Table 1 shows the testing scenario for single workload and Table 2 shows the comparison of both the Kernel in case of single workload.

TABLE I
Testing Scenario for Single Workload

S.No.	Benchmark Process	Number of Samples
1.	7zip	50
2.	Gzip	50
3.	Flac-audio	50
4.	Timed Linux Compilation	50
5.	RAM Speed SMP	50
6.	postmark	50
7.	Sunflow Rendering System	50

Note : Single sample is average of 3 - 5 sample records.

TABLE II
Comparison of both the Kernels in case of Single Workload

S.No.	BenchMark	Original Kernel 4.4.1	Modified Kernel 4.4.1
1.	7zip (MIPS)	12091.5	13113.6
2.	Gzip (Seconds)	33.75987	30.1221
3.	Flace-audio (Seconds)	11.612979	11.612
4.	Timed Linux Compilation (Seconds)	233.862	203.295
5.	RAM Speed SMP (MBPS)	6557.231	6553.287
6.	postmark (TPS)	202.92	188.5
7.	Sunflow Rendering System(Seconds)	5.646	5.618

Note : single sample is average of 3 - 5 sample records.

The multiple workload means providing the workload through multiple processes, they may be either CPU bound or IO bound or combination of both. For example 7zip, Flac-audio, Gzip, n-queen etc. These processes are available in phoronix Test Suite. The combination of different processes

TABLE III

TESTING SCENARIO FOR MULTIPLE WORKLOAD

S.No.	Scenario	Benchmark Process Set	Number of Samples
1.	set1	Timed Linux Compilation, flac-audio	50
2.	set2	Gzip, Sunflow Rendering System	50
3.	set3	Flac-audio, Sunflow Rendering System	50

TABLE IV

COMPARISON OF BOTH THE KERNELS IN CASE OF MULTIPLE WORKLOAD

S.No.	BenchMark	Original Kernel 4.4.1	Modified Kernel 4.4.1
1.	Timed Linux Compilation, flac-audio (Seconds)	257.066	256.712
2.	Gzip, Sunflow Rendering System (Seconds)	52.760	39.939
3.	Flac-audio, Sunflow Rendering System (Seconds)	20.406	19.849

are executed by Phoronix Test Suite on multicore processor. Table 3 presents the testing scenario for multiple workload and Table 4 presents the comparison of both the Kernel in case of multiple workload.

V. RESULTS

A. Comparison of Benchmarks Results with Original Kernel

The load imbalance among the cores can be reduced by distributing the equal load on all the cores. The table 5 shows the percentage change in the load imbalance on multicore processors for the given test scenarios for the average load balancer with respect to the existing load balancer.

$$\frac{ExistingKernelPer. - ProposedKernelPer.}{ExistingKernelPerformance} * 100$$

TABLE V

PERCENTAGE CHANGE IN LOAD IMBALANCE W.R.T. THE EXISTING LOAD BALANCER (SINGLE WORKLOAD)

Benchmark	Original Load Balancer	Avg Load Balancer	Percentage Improvement
7zip (MIPS)	12091.5	13113.6	7.79 %
Gzip (Seconds)	33.75987	30.1221	10.77 %
Flace-audio (Seconds)	11.612979	11.612	0.0084 %
Timed Linux Compilation (Seconds)	233.862	203.295	13.07 %
RAM Speed SMP (MBPS)	6557.23	6553.28	0.060 %
postmark (TPS)	202.92	188.5	7.10 %
Sunflow Rendering System (Seconds)	5.64	5.61	0.48 %

TABLE VI

PERCENTAGE CHANGE IN LOAD IMBALANCE W.R.T THE EXISTING LOAD BALANCER (MULTIPLE WORKLOAD)

Benchmark	Original load balancer	Avg Load Balancer	Percentage Improvement
set 1	257.066	256.712	.0013 %
set 2	42.760	39.939	6.59 %
set 3	20.406	19.849	2.73 %

Average percentage Improvement = 4.86057 %

VI. CONCLUSION AND FUTURE WORK

In the linux kernel development the most important work for better utilization of cores is load balancing. The load balancing aims to maintain fairness with each core, i.e. allocating proper workload to each core, but to maintain the fairness with each core we are required to optimize the existing load balancing approach. In the proposed work the Average Weight load balancer maintains the fainess with all cores. Results of the experimentation show that the performance is improved in terms of MIPS and execution time. The MIPS is increased for CPU bound processes and the execution time is reduced.

A. Future Direction

In the proposed approach we have performed the load balancing for homogenous multicore processors. This work can be extended for heterogeneous multicore processors. This will be a significant contribution, as processors with heterogenous cores are being designed now and their use will increase in future.

REFERENCES

[1] Suchi Johari, Arvind kumar, *Algorithmic Approach for Applying Load Balancing During Task Migration in Multicore System*, 978-1-4799-7683-6/14@2014 IEEE

[2] Juan M. Cebrian, daniel Sanchez, Juan L. Aragon, Stefanos Kaxiras, *Efficient intercore power and thermal balancing for multi-core processors*, Springer computing(2013) 95:537-566 DOI 10.1007/s00607-012-0236-6

[3] Zhigang Sun, Rui Wang, Ludan Zhang, Qi Li, Liya Chen, Jin Wu. Yi Liu, *Cache-aware Scheduling for Energy Efficiency on Multi-Processors*, International Conference on Computer DistributedControl and Intelligent Enviromental Monitoring 2012.

[4] Josue Feliu, Julio Sahuquillo, Salvador Petit, and Jose Duato, *Understanding Cache Hierarchy Contention in CMPs to Improve Job Scheduling*, IEEE 26th International Parallel and Distributed Processing Symposium 2012 ,pp 1530-2075.

[5] Jejurika R, Gupta R. *Energy Aware Task Scheduling with Task Synchronization for Embedded Real-time Systems*, [J]. IEEE Trans on Computer Aided Design o f Integrated Circuits and Systems, 2006, pp 1024-1037.

[6] Muhammad Zakarya, Nadia Dilawar, Naveed Khan , *A Survey on Energy Efficient Load Balancing Algorithms over Multicores* , International Journal of Research in Computer Applications & Information Technology Volume 1, Issue 1, July-September, 2013, pp. 59-67, IASTER 2013 www.iaster.com, ISSN Online: 2347-5099, Print: 2348-0009

[7] Geunsik Lim, Changwoo Min, YoungIk Eom, *Load-Balancing for Improving User Responsiveness on Multicore Embedded System*, IT R&D program of MKE/KEIT [10041244, SmartTV 2.0 Software Platform].

[8] D. Bautista, J. Sahuquillo, H. Hassan, S. Petit, J. Duato, *A Simple Power-Aware Scheduling for Multicore Systems when Running Real-Time Applications*, Department of Computer Engineering (DISCA) Universidad Politecnica de Valencia, Spain