International Journal of Computer Information Systems and Industrial Management Applications.
ISSN 2150-7988 Volume 5 (2013) pp. 557-563

Network Traffic Monitoring and Control for Multi core processors in cloud computing applications

A.S.Radhamani[1], E.Baburaj[2]

[1]Department of Computer Science and Engineering, Manonmanium Sundaranar University,
Research Scholar, Tirunelveli.
asradhamani@gmail.com

[2]Department of Computer Science and Engineering, Sun College of Engineering and Technology,
Professor , Nagercoil
alanchybabu@gmail.com

***Abstract*- Parallel Programming (PP) used to be an area is confined to scientific and cloud computing applications. However, with the proliferation of multicore processors, parallel programming has definitely become a mainstream of concern. To satisfy the requirement, one can leverage multi-core architectures to parallelize traffic monitoring so as to progress information processing capabilities over traditional uni-processor architectures. In this paper an effective scheduling framework for multi-core processors that strike a balance between control over the system and an effective network traffic control mechanism for high-performance computing is proposed. In the proposed Cache Fair Thread Scheduling (CFTS), information supplied by the user to guide threads scheduling and also, where necessary, gives the programmer fine control over thread placement. For this wait-free data structure are applied in lieu of conventional lock-based methods for accessing internal scheduler structure, alleviating to some extent serialization and hence the degree of contention. Cloud computing has recently received considerable attention, as a promising approach for delivering network traffic services by improving the utilization of data centre resources. The primary goal of scheduling framework is to improve application throughput and overall system utilization in cloud applications. The resultant aim of the framework is to improve fairness so that each thread continues to make good forward progress. The experimental results show that the parallel CFTS-WF could not only increase the processing rate, but also keep a well performance on stability which is important for cloud computing. This makes, it an effective network traffic control mechanism for cloud computing.**

Keywords: Cache Fair Thread Scheduling, multi-core, wait free data structure, cloud computing, network traffic

I. Introduction

Cloud computing and multicore processor architecture is two emerging classes of execution environments that are rapidly becoming widely adopted for the deployment of Web services. Explicit parallel architectures require specification of parallel task along with their interactions. In cloud computing environment multicore network traffic analysis become challenge for several reasons. First, packet capture applications are memory bound, but memory bandwidth does not seem to increase as fast as the number of core available [3]. Second, balancing the traffic among different processing units is challenging, as it is not possible to predict the nature of the incoming traffic. Exploiting the parallelism with general-purpose operating systems is even more difficult as they have not been designed for accelerating packet capture. During the last three decades, memory access has always been one of the worst cases of scalability and thus several solutions to this problem have been proposed .With the advent of Symmetric Multiprocessor Systems (SMP), multiple processors are connected to the same memory bus, hereby causing processors to compete for the same memory bandwidth. Integrating the memory controller inside the main processor is another approach for increasing the memory bandwidth. The main advantage of this architecture is fairly obvious: multiple memory modules can be attached to each processor, thus increasing bandwidth. In shared memory multiprocessors, such as SMP and NUMA (Non Uniform Memory Access), a cache coherence protocol [19] must be used in order to guarantee synchronization among processors. A multi-core processor is a processing system composed of two or more individual processors, called cores, integrated onto a single chip package. As a result, the inter-core bandwidth of multi-core processors can be many times greater than the one of SMP systems.

For traffic monitoring and control applications, the most efficient approach to optimize the bandwidth utilization is to reduce the number of packet copies. In multi-core and multi-processor architectures, memory bandwidth can be wasted in many ways, including improper scheduling, wrong balancing of Interrupt ReQuests (IRQ) and subtle mechanisms such as false sharing [14]**.** For these reasons, squeezing performance out of those architectures require additional effort. Even though there is a lot of ongoing research in this area, most of the existing schedulers are unaware of architectural differences between cores; therefore the scheduling does not guarantee the best memory bandwidth utilization. This may happen when two threads using the same data set are scheduled on different processors, or on the same multi-core processors having separate cache domains. Therefore an effective software scheduler to better distribute the workload among threads can substantially increase the scalability is achieved by CFTS. In cloud computing software

services are provided in the cloud and not on the local computer. In a multicore processor, several processor cores are placed on the same chip. These cores can be utilized to either execute several independent programs at the same time or execute a single program faster. In the cloud, all the resources including infrastructure, platform and software are delivered as services, which are available to customers in a pay-peruse model. In this way, cloud computing gains advantages such as cost savings, high availability, and easy scalability [15]. Cloud must guarantee all resources as a service and provide them to users by means of customized System Level Architectures (SLAs). However, in the cloud, there might be tens of thousands or even more users accessing resource simultaneously, which give an extremely high pressure on the cloud. When all users are requiring service from the cloud, the out traffic from data center will be tremendous and the network bandwidth must be managed effectively to serve all users [23]. So, it is necessary to control the data streams and make a better utilization of free network resources in the cloud.

In this paper, a network traffic monitoring and control for multi core processors based on cloud is proposed. Also it monitors the application behavior at run-time, analyze the collected information, and optimize multi-core architectures for cloud computing resources. The traditional operations on traffic packet structures are modified to reduce the strong dependency in the sequential code. Then wait-free data structures are applied selectively to make multi-core parallelization easier and manageable. Based on this, the parallel network traffic controller can run in a 1- way 2-stage pipelined fashion on a multi-core processor, which not only increases the processing speed significantly, but also performs well in stability. The scheduling framework is implemented such that it first compares the existing deadline monotonic without wait free data structure and with wait free data structure. Similarly for parallel applications like cloud computing the CFTS is also compared with and without wait free data structure. The remainder of this paper is organized as follows. After the related work in Section 2, Section 3 gives description of the problem for multi core systems in cloud applications using Cache Fair Thread Scheduling algorithm and Section 4 describes problem evaluation and section 5 provides results and discussions, followed by conclusion.

II. Related Work

To characterize scientific and transactional applications in Cloud infrastructures - IaaS, identifying the best virtual machine configuration in terms of the optimal processor allocation for executing parallel and distributed applications are proposed in [6]. A self-organized architecture for a Cooperative Scheduling System considering that it must be able to perform scheduling in highly dynamic environments where there is incomplete information and changes often occur is implemented in [2].The effect of heterogeneous data on the scheduling mechanisms of the cloud technologies and a comparison of performance of the cloud technologies under virtual and nonvirtual hardware platforms are given in [7]. The number of cores which fit on a single chip is growing at an exponential rate while off-chip main memory bandwidth is growing at a linear rate at best. This core count to off-chip bandwidth disparity causes per-core memory bandwidth to decrease as process technology advances. An analytic model to study the tradeoffs of utilizing increased chip area for larger caches versus more cores is introduced in [1]. In [18], different scheduling policies for multicore processor with wait free data structure are evaluated in cloud computing applications. The idea of treating clouds and multi cores as a single computing environment has been introduced by DavidWentzlaff et al. within the contest of the Fos Operating System [20]. They propose the development of a modern Operating System to target at the same time and in parallel the two different computing platforms, using a well defined service based architecture. As cloud computing is a relatively new concept, it is still at the early stage of research. Most of the published works focuses on general description of cloud, such as its definition, advantages, challenges, and future [4] and [10]. In detail, security is a very popular and important research field in cloud computing. Some researches focus on the data confidentiality and integrity in cloud computing, and some analyze security problems in cloud computing from different points of view, such as network, servers, storage, system management and application layer [22]. Besides security, another hot topic in cloud computing is virtualization technologies. A security Private Virtual Infrastructure [PVI] and the architecture that cryptographically secures each virtual machine are proposed in [12]. A virtual machine image management system is introduced in [20], and a real-time protection solution for virtual machine is given in [5]. A Run-Time Monitor (RTM) which is a system software to monitor the characteristics of applications at run-time, analyze the collected information, and optimize resources on a cloud node which consists of multi-core processors are described in [16]. In this study on constructing many core architectures well suited for the emerging application space of cloud computing where many independent applications are consolidated onto a single chip is described. Nevertheless, both computing platforms require the software to correctly use a very large and potentially heterogeneous pool of available execution resources. For instance, in the near future multicore machines with more than 64 cores will become main stream [17]. On such a computing platform, the correct usage of each core will become a relevant issue not only affecting the overall performance of the service, but also impacting its power consumption.

To make CFTS capable for cloud computing, parallelization technology on multi-core processor is required. A research trend on multi-core parallelization is wait-free data structures. A general introduction to the wait free data structures is given in [8].

III. Problem Statement

This section first describes CFTS in detail by comparing it with static scheduling algorithm (deadline monotonic) for multi core systems while running cloud applications. The scheduling primitives must support a wide variety of parallelization requirements. Moreover, some applications need different scheduling strategies for different program phases. In deadline monotonic, the time that a program takes to run will not depend only on its computation requirements but also on the ones of its co-runners. Therefore, the scheduler not only must select the task to be launched (e.g., a critical task) but also the appropriate core. The core must be selected according to the computational requirements of the tasks already running in each core. Therefore it is measured as time

consuming for cloud applications. Also, as the number of tasks (traffic) increases, the static scheduling system employed by a traditional OS will no longer guarantee optimal execution of tasks. Therefore proposed CFTS reduces the effects of unequal CPU cache sharing that occur on these processors and cause unfair CPU sharing, priority inversion, and inadequate CPU accounting. CFTS attempts to minimize the effect of thread level data dependencies and maintain efficient execution of all threads in the processor without excessive overhead for scheduling computation. In our work thread scheduling performs in parallel with CPU operation by utilizing resources and thereby minimizing the amount of time spent in the OS. In order to schedule the CFTS threads effectively, it must have information regarding the current state of all threads currently executing in the processor. To do gain this knowledge wait free data structures are implemented to store threads and maintain information about their current status. Therefore on parallelizing CFTS by applying wait-free design principles can be used for the allocation and management of shared network resources among different classes of traffic streams with a wide range of performance requirements and traffic characteristics in high-speed packet-switched network architectures. The proposed CFTS maximizes the throughput, and can then be used for efficient bandwidth management and traffic control in order to achieve high utilization of network resources while maintaining the desired level of service for cloud computing by CFTS scheduling in multi core systems.

A. Description of Cache Fair Thread scheduling algorithm

CFTS is suitable for cloud computing for its idea of bandwidth borrowing. It can not only control the bandwidth of all users, guaranteeing that all users could be given different levels of basic service by their payment, but also make more effective usage of free resource and make a better user experience. In the cloud, there could be different kinds of leaf users, e.g. 2Mbps, 5Mbps, regarding different service levels. Because CFTS is a dynamic traffic control mechanism, classes can be added or removed dynamically. This makes CFTS scalable enough for cloud computing.

On real hardware, it is possible to run only a single task at once, so while that one task runs, the other tasks that are waiting for the CPU are at a disadvantage - the current task gets an unfair amount of CPU time. In CFTS this fairness imbalance is expressed and tracked via the per-task p->wait_runtime (nanosec-unit) value. "wait_runtime" is the amount of time the task should now run on the CPU for it to become completely fair and balanced. CFTS's task picking logic is based on this p->wait_runtime value and it is thus very simple: it always tries to run the task with the largest p->wait_runtime value. So CFTS always tries to split up CPU time between runnable tasks as close to 'ideal multitasking hardware' as possible. This algorithm redistributes CPU time to threads to account for unequal cache sharing: if a thread's performance decreases due to unequal cache sharing it gets more time, and vice versa. The challenge in implementing this algorithm is determining how a thread's performance is affected by unequal cache sharing using limited information from the hardware. The cache-fair scheduling algorithm does not establish a new CPU sharing policy but helps *enforce* existing policies. The key part of our algorithm is correctly computing the adjustment to the thread's CPU quantum. The given four-steps are used to compute the cache-fair scheduling algorithm adjustment.

1. Determine a thread's *fair L2 cache miss rate* – a miss rate that the thread would experience under equal cache sharing.
2. Compute the thread's *fair CPI rate* – the cycles per instruction under the fair cache miss rate.
3. Estimate the *fair number of instructions* – the number of instructions the thread would have completed under the existing scheduling policy if it ran at its fair CPI rate (divide the number of cycles by the fair CPI). Then measure the actual number of instructions completed.
4. Estimate how many CPU cycles to give or take away to compensate for the difference between the actual and the fair number of instructions. Adjust the thread's CPU quantum accordingly.

The algorithm works in two phases:

1. *Searching phase:*

The scheduler computes the fair L2 cache miss rate for each thread.

2. *Calibration phase:*

A single calibration consists of computing the adjustment to the thread's CPU quantum and then selecting a thread from the best effort class whose CPU quantum is adjusted to offset the adjustment to the cache-fair thread's quantum. Calibrations are repeated periodically.

The challenge in implementing this algorithm is that in order to correctly compute adjustments to the CPU quanta and need to determine a thread's fair CPI ratio using only limited information from hardware counters [13]. This algorithm reduces L2 contention by avoiding the simultaneous scheduling of problematic threads while still ensuring real-time constraints.

B. Description of Static Algorithm (Deadline Monotonic)

To meet hard deadlines implies constraints upon the way in which system resources are allocated at runtime. This includes both physical and logical resources. Conventionally, resource allocation is performed by scheduling algorithms whose purpose is to interleave the executions of processes in the system to achieve a pre-determined goal. For hard real-time systems the obvious goal is that no deadline is missed. One scheduling method that has been proposed for hard real-time systems is a type of deadline monotonic algorithm [11] .This is a static priority based algorithm for periodic processes in which the priority of a process is related to its period. With this algorithm, several useful properties, including a schedulability test that is sufficient and necessary the constraints that it imposes on the process system are severe: processes must be periodic, independent and have deadline equal to period. The processes to be scheduled are characterized by the following relationship:

Computation time < deadline < period

Based on this each core is characterized by:

1) The frequency of each core, fj, given in cycles per unit time. With DVS, fj can vary from $fj\ min$ to $fj\ max$, where $0 < fj\ min < fj\ max$. From frequency it is easy to obtain the speed of the core, Sj, which is simply the inverse of the frequency.

2) The specific architecture of a core, $A(corej)$, includes the type the core, its speed in GHz, I/O,
local cache and/or memory in Bytes.

Tasks: Consider a parallel application, $T = \{t1, t2, \ldots, tn\}$, where *ti* is a task. Each task is characterized by:
1) The computational cycles, *ci*, that it needs to complete. (The assumption here is that the *ci* is known *a priori*.)
2) The specific core architecture type, *A*(*ti*), that it needs to complete its execution.
3) The deadline, *di*, before each task has to complete its execution.

The application, *T*, also has a deadline, *D*, which is met if and only if the deadlines of all its tasks are met. Here, the deadline can be larger than the minimum execution time and represents the time that the user is willing to tolerate because of the performance-energy trade-offs. The number of computational cycles required by *ti* to execute on *corej* is a finite positive number, denoted by *cij*. The execution time of *ti* under a constant speed *Sij*, given in cycles per second is ,

$$tij = cij/Sij.$$

C. Description of Wait free data structure

A wait-free data structure is a *lock-free* data structure with the additional property that every thread accessing the data structure can make complete its operation within a bounded number of steps, regardless of the behaviour of other threads. Each thread is guaranteed to be progressed itself or a cooperative thread [9]. This property means that high-priority threads accessing the data structure never have to wait for low-priority threads to complete their operations on the data structure, and every thread will always be able to make progress when it is scheduled to run by the OS. For real-time or semi-real-time systems this can be an essential property, as the indefinite wait-periods of blocking or non-wait-free lock-free data structures do not allow their use within time-limited operations. A wait-free data structure has the maximum potential for true concurrent access, without the possibility of busy waits.

IV. Problem Evaluation

The advent of Cloud computing platforms and the growing pervasiveness of multicore processor architectures have revealed the inadequateness of traditional programming models based on sequential computations, opening up many challenges for research on parallel programming models for building distributed, service-oriented systems.
To investigate the effects of different scheduler configurations on the performance of multicore processor, implementation is a suite of MATLAB simulation. Parallel Computing Toolbox provides several high-level programming constructs that help us to convert our applications to take advantage of computers equipped with multi core processors. Parallel programming improves performance by breaking down a problem into smaller sub problems that are distributed to multiple processors. Thus, the benefits are two-fold. First, the total amount of computation performed by each individual processor is reduced, resulting in faster computation. Second, the size of the problem can be increased by using more memory available on multiple processors.
A parfor (parallel for) loop is useful in situations that require many loop iterations of a simple calculation, is used. As maxNumCompThreads() controls the parallelism of the multithreading approach, the matlabpool command controls the parallel behavior of the parfor syntax. matlabpool sets up a task-parallel execution environment in which parallel for-loops can be executed interactively from the MATLAB command prompt. The iterations of parfor loops are executed on labs (MATLAB sessions that communicate with each other). As a result, they can run on separate computers connected via a network. In the proposed work we only need to know that Parallel Computing Toolbox makes parfor work efficiently on a single multicore system.

```
EffThread=Nthread(index);
 s=[];
e=1;
t=0;
 limit=numel(EffThread);
  tStart=tic;
while(limit>e)
for k=1:CacheMemory
   for i=1:NrMultiCore
            if(limit<=e)
          break;
      end
         t=EffThread(e);
      e=e+1;
            if(limit<=e)
          break;
      end
          if(i>1)
       t1=EffThread(e);
             if(t==t1)
         s(k,i)=EffThread(e);%same core
         pause(.002)
       else
          s(k,i)=EffThread(e);%Other cores
          pause(.002)
            end
 end
    end
  % pause(.05);
end
if(limit<=e)
   break;
end
 pause(.12)
   end
te=toc(tStart);
```

Table 1 .CFTS Algorithm

The main function of the deadline monotonic scheduling phase is to coordinate the execution order of tasks. Based on this the execution priority assignments are assigned by each task and are arranged in the descending order, and then be executed.
For CFTS in the implementation it is assumed that the a multicore platform consisting of M cores and A cache partitions, and a set of τ independent tasks whose numbers of cache partitions (cache space size needed) and Ei , WCET(Worst Case Execution Time) are known for the platform , and further it is assumed that τi = (Ai,Ei, Di,Ti) to denote a task where Ai is the cache space size, Ei is the worst-case execution time (WCET), Di ≤ Ti is the relative deadline for each release, and Ti is the minimum inter-arrival separation time also referred to as the period of the task. We further assumed that all tasks are ordered by priorities, i.e., τi

has higher priority than τj iff i<j. The utilization of a task τi is Ui = Ei/Ti and its relaxation Ri = Di - Ei, which is the longest delay allowed before actually running without missing its deadline. In the system design phase, one can adjust tasks L2 cache space sizes (and therefore their WCETs) to improve the system real-time performance, which can be built upon the schedulability analysis techniques. Also in the CFTS algorithm at any time, at any two running tasks cache spaces are non-overlapped. A task can get to execute only if it gets an idle core as well as enough space (not necessarily continuous) on the shared cache. The first technical contribution is a sufficient schedulability test for multicores with shared L2 cache, for eight different applications (coop, imrec, ood, games, plros, zip, wpro, vlsi). To improve its scalability, second schedulability test for network traffic in clouds is implemented. To evaluate the performance and scalability of our techniques, we use randomly generated task sets. The following part of Matlab code that describes the effective thread, cache and core allocation.

Typically, a separate set of performance counters is available for each core, and can be programmed to track events originating from that core. In this implementation a counter is programmed at each core to track lower-level (shared) cache misses. Since jobs execute sequentially, one can measure the number of cache misses incurred for a job by resetting the counter to zero at the start of execution, and recording the total misses observed by the counter upon completion. The observed misses can then be used to calculate a per-job ET estimate. Computed ET estimates are cached to minimize computation.

V. Results and Discussion

In the proposed work the time that it takes to complete its work segments in the Cache Fair Thread Scheduler with Wait Free data structures (CFTS-WF) and Cache Fair Thread Scheduler (CFTS) schedules are compared. This quantity is referred to as *completion time*. When running with a static scheduler, the difference between completion times is larger, but when running with the cache fair scheduler, it is significantly smaller. Figure 1 demonstrates normalized completion times with the static scheduler and Figure 2 demonstrates normalized completion times with the Cache Fair Thread scheduler. Figure 3 shows the Performance variability of different types of schedulers with variations in the cache and core sizes for the different applications and the table shows the average completion time for different applications with various schedulers. From the performance analysis and comparison made it is clear that, CFTS-WF provides maximum speed up, load balancing among all scheduling strategies used to obtain a minimum processing time.

Hence the CFTS-WF seek to maximize the use of concurrency by mapping independent tasks on different threads, so that to minimize the total completion time by ensuring that processes are available to execute the tasks on the critical path as soon as such tasks become executable, and it should seek to minimize interaction among processes by mapping tasks with a high degree of mutual interaction onto the same process.

Static Scheduler

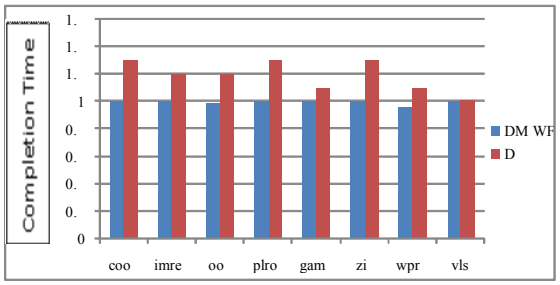

Figure 1 Performance Variability with Static scheduler

Cache Fair Thread scheduler

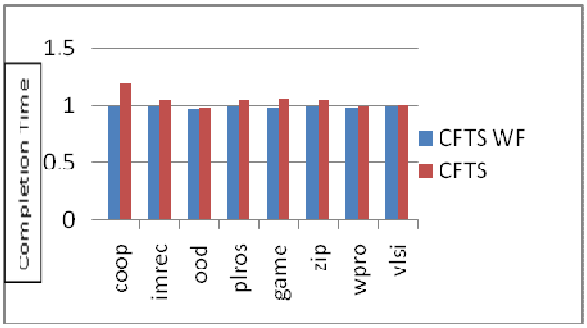

Figure 2 Performance variability with Cache fair Thread Scheduler

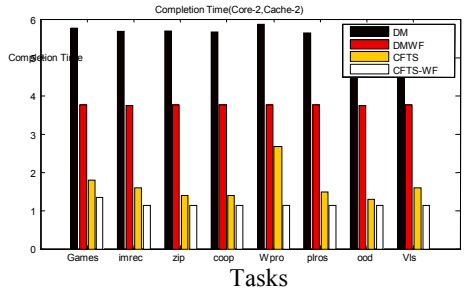

Figure 3(a). Performance Variability with core-2, cache-2

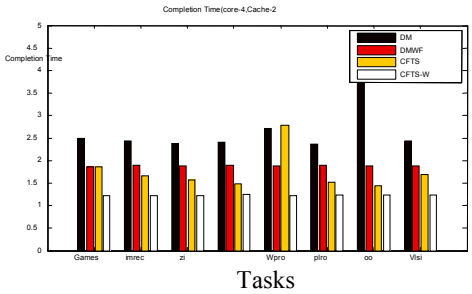

Figure 3(b). Performance Variability with core-4, cache-2

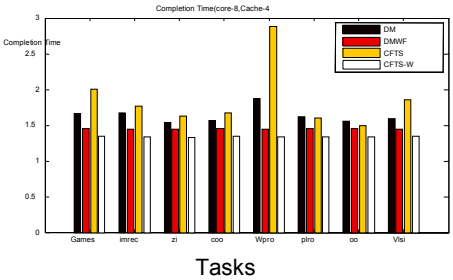

Figure 3(c). Performance Variability with core-8, cache-4

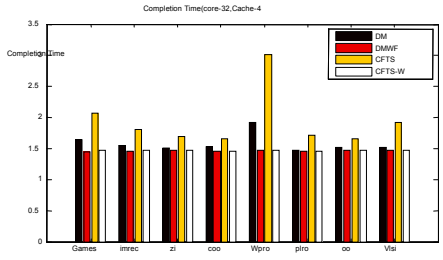

Figure 3(d). Performance Variability with core-32,cache -4

Algorithm	Core2, cache 2 (Average completion Time in Sec)	Core4, cache 4 (Average completion Time in Sec)	Core8, cache 4 (Average completion Time in Sec)	Core32, cache 4 (Average completion Time in Sec)
DM	5.9	2.6	1.7	1.6
DMWF	3.8	1.9	1.4	1.5
CFTS	1.9	1.7	2.0	1.7
CFTSWF	1.3	1.2	1.3	1.4

Table 2. Average Completion time (in sec) for different Algorithms

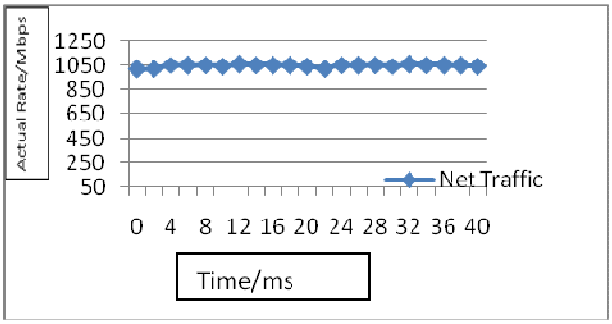

Figure 4. Output traffic rate of total traffic

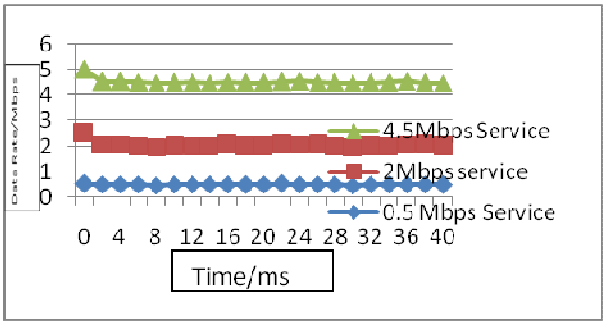

Figure 5 .Output traffic rate of a selected user

It is again designed to test whether the wait –free based CFTS could be competent for cloud scenario described in section 3 under extremely high traffic pressure. Figure 4 and Figure 5 shows the output rate sampling at a certain time interval for both the total CFTS and a random selected user. To make the results more accurate, different time interval for total traffic and user traffic is used, because their rates are at different levels. It is understood that the wait-free FIFO based CFTS-WF performs traffic control quite stable, and the resulting rate lines are nearly smooth, indicating the traffic rates are accurately retained. The stability is important for cloud computing because it might face tens of thousands users at the same time.

VI. Conclusion

A significant body of the performance modeling research literature has focused on various aspects of the parallel computer scheduling problem and the allocation of computing resources among the parallel jobs submitted for execution. Several classes of scheduling strategies have been proposed for such computing environments, each differing in the way the parallel resources are shared among the jobs. This includes the class of space-sharing strategies that share the processors in space by partitioning them among different parallel jobs, the class of time-sharing strategies that share the processors by rotating them among a set of jobs in time, and the class of scheduling strategies that combine both space-sharing and time-sharing. In this paper, a wait free based parallel CFTS is implemented for effective and stable traffic control in the cloud. Based on the algorithms on accessing data structures and the usage of wait free FIFO, the parallel CFTS can run a pipelined fashion. The experimental analysis and evaluation results both indicate that parallel CFTS WF is more suitable for cloud computing, due to its excellent performance on both line rate and stability. Moreover, parallel network application based on multi-core processor is cheaper and more scalable than special hardware and explore its more effective usage in cloud computing. This scheduler is a good solution to achieve the best VM (Virtual Machine) environment for running applications under Cloud Computing environments. In future CFTS-WF can be experimented can be executed under a cluster with more processor scalability to allocate and deallocate cores in the VM in heterogeneous communication and computation environment.

References

[1] Agarwal, Anant; Miller, Jason; Beckmann, Nathan; Wentzlaff, David," Core Count vs Cache Size for Manycore Architectures in the Cloud", *CSAIL Technical Reports*, Volume 6568/2011, pp.39-50.

[2] Ana Madureira and Ivo Pereira, "Intelligent Bio-Inspired System for Manufacturing Scheduling under Uncertainties" *International Journal of Computer Information Systems and Industrial Management Applications*, Volume 3 (2011) pp.072-079

[3] K. Asanovic and others, The landscape of parallel computing research: A view from Berkley. *Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkley*, December (2006).

[4] R. Buyya, "Market-Oriented Cloud Computing: Vision, Hype, and Reality of Delivering Computing as the 5th Utility", *Proc. of 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'09), Shanghai, China*, May, 2009, pp. 1-6, doi: 10.1109/CCGRID.2009.97.

[5] M. Christodorescu, R. Sailer. D. L. Schales, D. Sgandurra and D. Zamboni, "Cloud Security Is Not (Just) Virtualization Security", *Proc. of the ACM workshop on Cloud computing security 2009,*

Chicago, IL,US, 2009, pp.97-102, doi: 10.1145/1655008.1655022.

[6] Denis R. Ogura, Edson T. Midorikawa, "Characterization of Scientific and Transactional Applications under Multi-core Architectures on Cloud Computing Environment," *IEEE International Conference on Computational Science and Engineering*, pp. 314-320, 2010.

[7] Ekanayake, J.; Gunarathne, T.; Qiu, J," Cloud Technologies for Bioinformatics Applications" , *IEEE Transactions on Parallel and Distributed Systems,* Volume: 22,pp 998 – 1011.

[8] K. Fraser and T. Harris, "Concurrent programming without locks", *ACM Transactions on Computer Systems*, vol. 25 (2), May 2007.

[9] J. Giacomoni, T. Moseley, and M. Vachharajani, "Fastforward for efficient pipeline parallelism: A cache-optimized concurrent lock free queue", *Proc. of PPoPP'08, New York, NY, USA*, February 2008, pp.43-52.

[10] J. Heiser and M. Nicolett, "Accessing the Security Risks of Cloud Computing", *Gartner Inc., Stanford, CT,2008,http://www.gartner.com/.*

[11] Ishfaq Ahmad, Sanjay Ranka, Sanjay Ranka, "Using Game Theory for Scheduling Tasks on Multi- Core Processors for Simultaneous Optimization of Performance and Energy", 2008.pp. 1-6.

[12] F.J. Krautheim, "Private Virtual Infrastructure for Cloud Computing", in *HotCloud'09, San Diego, CA, USA*, June, 2009.

[13] S. Kim, D. Chandra and Y. Solihin. "Fair Cache Sharing and Partitioning in a Chip Multiprocessor Architecture", *In Intl. Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2004,pp.111-122

[14] C. Leiserson and I. Mirman, "How to Survive the Multi-core Software Revolution", *Cilk Arts*, (2009).

[15] Lizhe Wang, Jie Tao, Gregor von Laszewski, Holger Marten," Multicores in Cloud Computing: Research Challenges for Applications", *Journal of computers,* vol. 5, no. 6, june 2010.

[16] Mikyung Kang , Dong-In Kang, Stephen P. Crago, Gyung-Leen Park and Junghoon Lee , **"**Design and Development of a Run-Time Monitor for Multi-Core Architectures in Cloud Computing" *, Sensors 2011*, pp. 3595-3610.

[17] Patterson. "The Trouble with Multi-core". *Spectrum, IEEE*, 47(7):pp.28–32, 2010.

[18] A.S.Radhamani and E.Baburaj ," Implementation of Cache Fair Thread Scheduling for multi core processors using wait free data structures in cloud computing applications", *Proc of 2011 World Congress on Information and Communication Technologies* , ©2011, IEEE,pp.604-609.

[19] T. Tartalja and V. Milutinovich, "The Cache Coherence Problem in Shared-Memory Multiprocessors", *Software Solutions*, ISBN: 978-0-8186-7096-1, (1996).

[20] D. Wentzlaff and A. Agarwal. "Factored Operating Systems (FOS): the Case for a Scalable Operating System for Multicores". *ACM SIGOPS Operating Systems Review*, 43(2):76–85, 2009.(10)

[21] J. Wei, X. Zhang, G. Ammons, V. Bala, and P. Ning, "Managing security of virtual machine images in a cloud environment", *Proc. Of the ACM workshop on Cloud computing security 2009, Chicago, IL,US*, 2009, pp.91-96, doi: 10.1145/1655008.1655021.

[22] M. Yildiz, J. Abawajy, T. Ercan, and A. Bernoth, "A Layered security Approach for Cloud Computing Infrastructure", *Proc. of the 10th International Symposium on Pervasive Systems, Algorithms, and Networks*, 2009, pp.763-767, doi: 10.1109/I-SPAN.2009.157.

[23] Zheng Li, Nenghai Yu, Zhuo Hao," A Novel Parallel Traffic Control Mechanism for Cloud Computing, *2nd IEEE International Conference on Cloud Computing Technology and Science,2010*.pp.376-382

Author Biographies

A.S.Radhamani has received her Under Graduate degree in Electronics and Communication Engineering from Bharathiyar University and Master's Degree in Computer Science and Engineering from Manonmanium Sundaranar University in 1995 and 2004 respectively. She is the author of six publications. She is currently pursuing her PhD Degree at MSU, Tirunelveli. She is an active researcher and her research interest includes multi core computing, parallel and distributed processing and cloud computing.

E. Baburaj has received his Under Graduate and Post Graduate in Computer Science and Engineering from Madurai Kamaraj Univesity. He has obtained his Doctoral Degree in Computer Science and Engineering from Anna University Chennai. Currently, he is the Dean of PG studies and Research of the Computer Science and Engineering Department, SCET, Nagercoil. His main research focuses on High Performance and Computer Networks. More than 20 publications are credited to his name.