Article can be accessed online at http://www.publishingindia.com

Scheduling Work Load Based on Priority in Cloud

Sindhuja R.*, Santhosh R.**

Abstract

Cloud computing offers ability to provide parallel and distributed simulated services remotely to the users through the internet. Services hosted within the "cloud "can potentially incur processing delay due to load sharing among other active services , and can cause active optimistic simulation protocols to perform poorly. Number of complex application runs in remote data centres, parallel processing capabilities often show a increase in utilization of CPU resources as parallelism grows, mainly because of communication and synchronization. To achieve certain level of utilization, Our proposed method partitions a node's computing capacity into the 4-tiers with low CPU priority, medium CPU priority, high CPU priority and very high CPU priority. In large datacenter, processes of a job may need to be allocated to nodes that are close to each other to minimize the communication cost. We provide scheduling algorithms for parallel jobs to make efficient use of the k-tiers VMs to improve the responsiveness of these jobs. We focus on improving resource utilization for datacenters that run parallel jobs; particularly we intend to make use of the remaining computing capacity of datacenter nodes that run parallel processes with low resource utilization to improve the performance of parallel job scheduling. The method is practical and effective for consolidating parallel workload in data centres.

Keywords: Distributed Computing, Parallel Computing, Parallel Simulation, Resource Consolidation, Scheduling, Virtualization

1. INTRODUCTION

Cloud computing techniques are used to share resources. It transfers the application software and database to the centralized large data Arash Ghorbannia Delavar, 2011 centres. Internet based online service do provide huge amount of storage space and customizable computing resources, this computing platform shift, however is eliminating the responsibility of local machines for data maintenance at the same time. Users are at the mercy of their cloud services providers (CSP) for the Utkarsh Pawde, 2011 availability and integrity of their data. Cloud storage allows users to remotely store their data and enjoy the on-demand high quality cloud application without the burden Idawaty Ahmad, 2008 of local hardware and software management. Though the benefits are clear, such a service is also relinquishing users' physical possession of their outsourced data, which inevitably poses new security risks towards the correctness of the data in cloud.

The latest emergence of cloud computing is a significant step towards realizing this utility computing model since it is heavily driven by industry vendors. Cloud computing Utkarsh Pawde, 2011 promises to deliver reliable services through next generation data centres built on virtualized compute and storage technologies. Users will be able to access applications and data from a "Cloud" anywhere in the world on demand and pay based on what they use.

2. BACKGROUND REVIEW

In this proposal, we focus on some parameters which are effective in a good scheduling. In the cloud computing environment, there are data-centre system that is assumed to collect and save the information. Gang scheduling Y. Lin , 1992 is an efficient technique for scheduling parallel jobs. Performance and cost while integrating mechanisms for job migration Y. Zhang, 2003 and handling of job starvation is low in this technique. The number of virtual machine (VMs) available at any moment is dynamic and scales according to the demands of the jobs being serviced. The aforementioned model is studied through simulation in order to analyse the performance and overall cost of

* PG Scholar, Department of Computer Science and Engineering, Karpagam University, Coimbatore, Tamil Nadu, India. E-mail: r.sindhuja2007@gmail.com

** Assistant Professor, Department of Computer Science and Engineering, Karpagam University, Coimbatore, Tamil Nadu, India. E-mail: santhoshrd@gmail.com

Gang scheduling with migration and consolidation Y. Zhang, 2003 for handling the starvation while executing the parallel jobs.

The features in this approach are :

1. Tasks are entered in system, each time.
2. All tasks are independent.
3. Each resources can process more than one request.

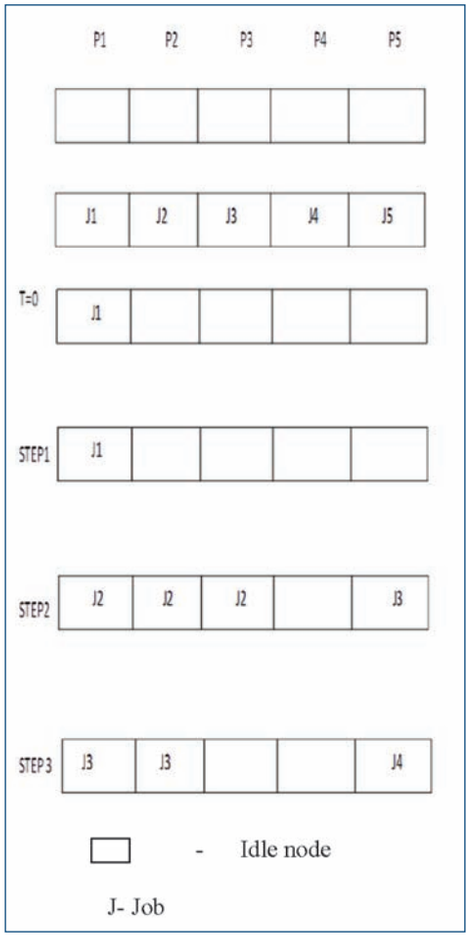
In server Consolidation, instead of small servers, a large server is utilized to increase the exploitation of exclusive hardware resources, decreasing the burning of energy and emission of CO2 D. Tsafrir, 2005 where as Virtualization is the primary approach for forwarding key to rising the business computing requirements. Data centre virtualization has been effect on four areas server hardware, operating system, storage, networks and application infrastructure.

Gang scheduling Y. Lin , 1992 Y. Wiseman , 2003 allows resource sharing among multiple parallel jobs. The computing capacity of a node is divided into time slices for sharing among the processes of jobs. The gang scheduling algorithm manages to make all the processes of a job progress together so that one process will not be in sleep state when another process needs to communicate with it. Backfilling Y. Zhang, 2003 is a technique that allows short/small jobs to use idle nodes while the job at the head of the queue does not have enough number of nodes to run. Backfilling can improve node utilization, but it requires each job to specify its maximum execution time so that only jobs that will not delay the start of the job at the head of the queue are backfilled Y. Zhang, 2003 . Here we considered migration backfilling (MBF) and compared with aggressive migration backfilling.

3. PARALLEL JOB SCHEDULING

We focus on improving resource utilization for data centres that run parallel jobs, particularly we intend to make use of the remaining computing capacity of data centre nodes that run parallel processes with low resource utilization to improve the performance of parallel job scheduling. Each job specifies the number of nodes required and the scheduler processes jobs according to the order of their arrival Y. Zhang, 2003 J. Jann, 1997. When there is a sufficient number of nodes to process the job at the head of the queue, the scheduler dispatches the job to run on these nodes; otherwise, it waits till jobs currently running finish and release enough nodes for the job. FCFS may cause node fragmentation and methods such as backfilling and Gang scheduling Y. Zhang, 2003 were proposed to improve it. However they do not target on the utilization degradation caused by parallelization itself.

Figure 1. Job Activation Back Filling



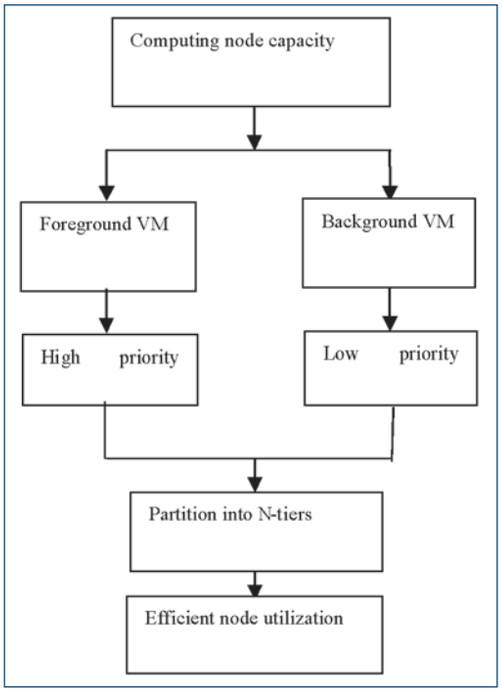
4. K-TIER CONSOLIDATION AND BACK FILLING ALGORITHM

To improve the utilization of servers allocated to the jobs, preserve FCFS order of jobs Utkarsh Pawde, 2011. This is the systematic way to consolidation parallel jobs.

To achieve two common goals,

- System Utilization
- Job responsiveness

Figure 2. Efficient Job Responsiveness



KTCBF - On the arrival of a new job

Input: M: the job allocation map; j: the new job;
Output: M': the updated job allocation map;
begin
add j to Q;
N_j<— the number of nodes needed by j;
N_{L_idle}<— the number of nodes with idle of Low_CPU VM;
N_{M_idle}<— the number of nodes with idle of Medium_CPU VM;
N_{H_idle}<— the number of nodes with idle of High_CPU VM;
N_{VH_idle}<— the number of nodes with idle of VeryHigh_CPU VM;
If $N_j \leq N_{L_idle}$then
remove j from Q;
dispatch (Low_CPU, j);
return;
if $N_j \leq N_{M\_idle}$ then
remove j from Q;
dispatch(Medium_CPU, j);
return;
if$N_j \leq N_{H_idle}$ then
remove j from Q;
dispatch (High_CPU, j);
return;
if$N_j \leq N_{VH_idle}$ then
remove j from Q;
dispatch (VeryHigh_CPU, j);
return;

CMCBF - On the depature of a Low_CPU job

input: Q: the incoming job queue ;
J_{High_CPU}: a list of jobs running in the High_CPU;
J_{Medium_CPU}: a list of jobs running in the Medium_CPU;
$J_{Very\_High\_CPU}$: a list of jobs running in the VeryHigh_CPU;
M: the job allocation map;
output: M': the updated job allocation map;
begin
remove the depature job from J_{High_CPU}; J_{Medium_CPU}; $J_{Very\_High\_CPU}$;
for each job j ∈ Q do
N_j<— the number of nodes needed by j;
N_{L_idle}<— the number of nodes with idle Low_CPU VM;
ifN_j<N_{L_idle} then
remove j from Q;
dispatch(Low_CPU,j);

The algorithm assumes that the state of a job can be saved and restored; therefore, the scheduler is able to suspend a job and resume it on other nodes in a later time. CMBF schedules jobs to run according to their arrival time when there is enough number of nodes. When the number of idle nodes is not sufficient for a job, another job with a later arrival time but smaller node number requirement may be scheduled to run via backfilling.

CMCBF - On the depature of a Medium_CPU job

input: Q: the incoming job queue ;
J_{Low_CPU}: a list of jobs running in the Low_CPU;
J_{High_CPU}: a list of jobs running in the High_CPU;
$J_{Very\_High\_CPU}$: a list of jobs running in the VeryHigh_CPU;
M: the job allocation map;
output: M': the updated job allocation map;
begin
remove the depature job from J_{Low_CPU}; J_{High_CPU}; $J_{Very\_High\_CPU}$;
for each job j ∈ Q do
N_j<— the number of nodes needed by j;

N_{M_idle}<— the number of nodes with idle Medium_CPU VM;
ifN_j<N_{M_idle} then
remove j from Q;
dispatch(Medium_CPU,j);

A preempted job is scheduled to run whenever it sees the total number of nodes that are either idle or occupied by jobs with a later arrival time is equal or greater than the number of nodes it needs. The job may preempt jobs arriving later but being scheduled on some nodes. The scheduler instructs these jobs to save states, suspends their execution, and moves them back to the job queue.

CMCBF - On the depature of a High_CPU job

input: Q: the incoming job queue ;
J_{Low_CPU}: a list of jobs running in the Low_CPU;
J_{High_CPU}: a list of jobs running in the High_CPU;
J_{Medium_CPU}: a list of jobs running in the Medium_CPU;
M: the job allocation map;
output: M': the updated job allocation map;
begin
remove the depature job from J_{Low_CPU}; J_{High_CPU}; J_{Medium_CPU};
for each job j ∈ Q do
N_j<— the number of nodes needed by j;
N_{VH_idle}<— the number of nodes with idle VeryHigh_CPU VM;
ifN_j<N_{VH_idle} then
remove j from Q;
dispatch(VeryHigh_CPU,j);

It ensures that a job is dispatched to run in foreground VMs whenever the number of foreground VMs that are either idle or occupied by jobs arriving later than it satisfies its node requirement. Meanwhile, it allows jobs to run in background VMs simultaneously with those foreground VMs to improve node utilization. CMCBF only dispatches a job to run in background VMs when the corresponding foregrounds VMs have utilization lower than a given threshold.

CMCBF- On the departure of a VeryHigh_CPU job

Input: Q: the throats job queue;
JVeryHigh_CPU:list of jobs running in the High_CPU;
M: the job allocation map;
Output M'; the updated job allocation map;
begin
j get the first job from Q ;
while j do
the number of nodes required by j;
the number of idle nodes VeryHigh_CPU VM;
if then
the number of nodes running jobs arriving later than j;
if then
if then
switch j to the Low_CPU or Medium_CPU or High_CPU tier save and suspend backing VeryHigh_CPU jobs of j, according to the descending order of their arrival time until N_{VH_idle}
if j then
remove j from J_{Low_CPU}, J_{Medium_CPU}, J_{High_CPU}
save the state of j and suspend its execution on the Low_CPU or Medium_CPU or High_CPU;
else
remove j from Q;
dispatch(VeryHigh_CPU,j);
j get the first job from Q;
;
function dispatch(flag,j)
begin
Sort the parallel processes of j in descending order of their node utilization based on the profile of this type of jobs;
if flag=VeryHigh_CPU then
sorted idle nodes in ascending order of their utilization (caused by the Low_CPU, Medium_CPU, High_CPU load);
for each process ji in the sorted list do
place ji to pi and run ji in the VeryHigh_CPU VM;
evict the Low_CPU, Medium_CPU, High_CPU job if the utilization of ji is above the threshold;
else
sorted idle nodes in ascending order of their utilization (caused by the VeryHigh_CPU load);
for each process ji do
place ji to pi and run ji in the Low_CPU, Medium_CPU, High_CPU VM if pi below the utilization threshold;
add j to J_{Low_CPU} or J_{Medium_CPU} or J_{High_CPU}

We extend the basic algorithms to be node utilization aware in attempting to improve the overall node utilization

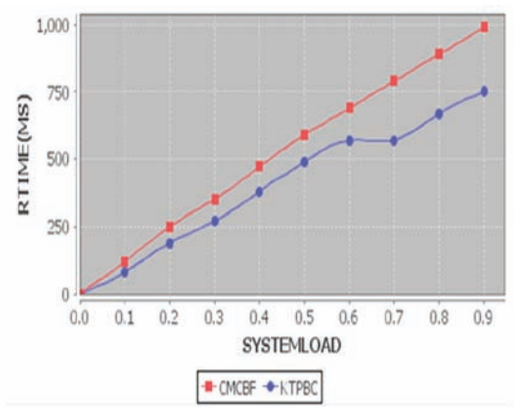
in the cloud. Based on our observation, we divide the computing capacity of a physical node into four tiers, namely low CPU priority, medium CPU priority, high CPU priority, very high CPU priority.We assume that a physical node can run at most two VMs with one in the low CPU priority, medium CPU priority, high CPU priority, very high CPU priority. In the following, we give a scheduling algorithm to handle four types of VM resources.

5. SIMULATION RESULT

CloudSim provides a generalized and extensible simulation framework that enables modeling, simulation and experimentation of emerging Cloud computation infrastructures and application services. Support for modeling and simulation of large scale Cloud computing data centers. It provides support for dynamic insertion of simulation elements, stop and resume of simulation. The result here is obtained using 50 virtual machines and 35 number of cloudlets and the performance has been shown in the graph.VMs have utilization lower than a given threshold. The foreground VM utilization can be obtained from the profile of foreground jobs, or from the runtime monitoring data. Collocate a background VM with which a foreground VM is determined through a simple process G. D'Angelo, 2011 . The process matches the background VM that is likely to incur high node utilization to the foreground VM that is likely to incur low node utilization.

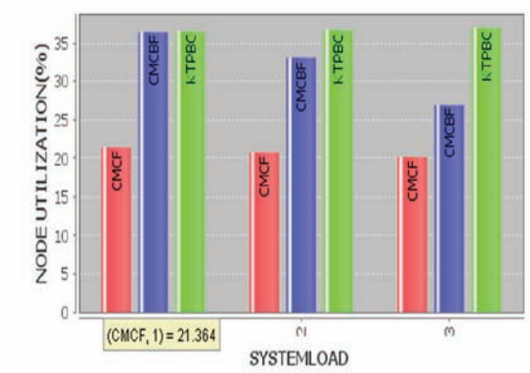
Figure 3 shows Average Response Time per Request (RTME): This Metric evaluates the Interval between when a resource allocation request is received, and when the response is sent. The response time is calculated as the sum of the times needed to pass through the four layers, and depends on the request.

Figure 3. Job Responsiveness



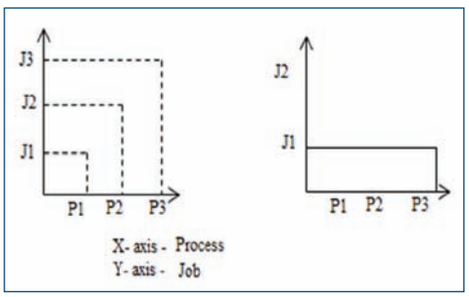
In figure 4 ,We study the impact of the accuracy of CPU usage information on the performance of each job. In addition, job migration cost and the amount of remaining computing capacity on each node also have impact on the performance of our scheduling algorithm. Job scheduling performance improves as the accuracy of CPU usage estimation increases.

Figure 4. Node Utilization



In figure 5: A preempted job is scheduled to run whenever it sees the total number of nodes that are either idle or occupied by jobs with a later arrival time is equal or greater than the number of nodes it needs. The job may preempt jobs arriving later but being scheduled on some nodes Idawaty Ahmad, 2008.

Figure 5. Allocation of Process and Job

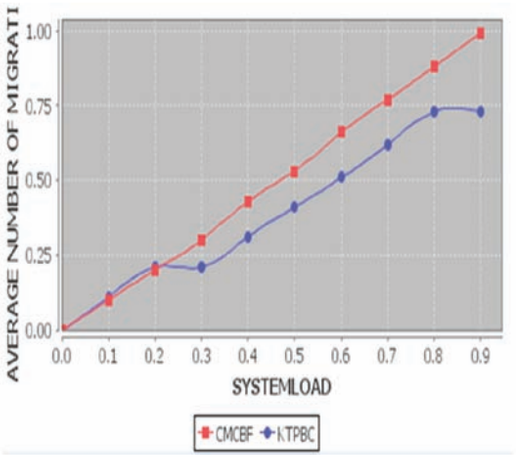


The scheduler instructs these jobs to save states, suspends their execution, and moves them back to the job queue. At time 0, job J1, J2, and J3 are allocated to five nodes and run in foreground VMs according to Algorithm 4. As J1 is a single-process job, therefore P1 cannot accommodate another VM running in background. However, J4 and J5 can run in background VMs at node P2-P5. How to collocate a background VM with which a foreground VM is determined through a simple process. The process matches the background VM that is likely to incur high node utilization to the foreground VM that is likely to incur

low node utilization. The process is shown in the dispatch function in Algorithm 2. This matchmaking process balances the load on physical nodes and minimizes the interference between background and foreground jobs.

The figure 6 represents the number of task migration required for algorithm. The average number of job migrations in K-tier is slightly less than that in Consolidation Migration Back Filling Y. Zhang, 2003 . Generally, the two algorithms with relaxed job execution order often require less number of job migrations than those preserving the order. This is due to that the computing capacity left by the any tier VMs allows many short jobs to be accommodated in the suitable tier VMs without incurring job migration.

Figure 6. Migration Graph



6. CONCLUSION

Due to the difficulty in realizing parallelism, many parallel applications show a pattern of decreasing resource utilization along with the increase of parallelism. Effectively partition the computing capacity of a datacenter node into k-tiers, which may further improve the node utilization Y. Lin , 1992 and responsiveness for parallel workload in the cloud. Our method partitions a node's computing capacity into the 4-tiers, low CPU priority, medium CPU priority, high CPU priority, very high CPU priority. The performance of jobs running in suitable VMs is close to that of jobs running in dedicated nodes. In large datacenter, processes of a job may need to be allocated to nodes that are close to each other to minimize the communication cost. We provide scheduling algorithms for parallel jobs to make efficient use of the k-tiers VMs to improve the responsiveness of these jobs. We focus on improving resource utilization for datacenters that run parallel jobs; particularly we intend to make use of the remaining computing capacity of datacenter nodes that run parallel processes with low resource utilization to improve the performance of parallel job scheduling. They can simultaneously process different jobs. The background job can therefore use the under utilized computing capacity whenever the foreground job cannot fully use it. Our method supports for k-tiers setting. Priority-based VM collocation incurs trivial performance impact to jobs running in the high-priority VMs. We provide a priority-based workload consolidation method to schedule parallel jobs in data centers to make use of underutilized node computing capacity to improve responsiveness.

REFERENCE

[1] Delavar, A. G. & Aryan, Y. (2011). A Synthetic heuristic algorithm for independent task scheduling in cloud systems. *IJCSI International Journal of Computer Science*, November, 8(6), 1694-814.

[2] Ingole, A., Chavan, S. & Pawde, U. (2011). *An Optimized Algorithm for Task Scheduling based on Activity based Costing in Cloud Computing*. 2nd National Conference on Information and Communication Technology (NCICT) 2011 Proceedings published in International Journal of Computer Applications® (IJCA).

[3] Rasooli, A. & Down, D. G. (2011). *An Adaptive Scheduling Algorithm for Dynamic Heterogeneous Hadoop Systems*. Proceeding CASCON '11 Proceedings of the 2011 Conference of the Center for Advanced Studies on Collaborative Research on IBM corp.USA.

[4] Gupta, B. D. & Palis, M. A. (2001). Online real-time preemptive scheduling of jobs with deadlines. *Journal of Scheduling*, November/December, 4(6), 297-312.

[5] D'Angelo, G. (2011). *Parallel and Distributed Simulation from Many Cores to the Public Cloud*. Proceedings of International Conference on High Performance Computing and Simulation (HPCS) (pp. 14-23).

[6] Ahmad, I, Shamala, S., Othman†, M. & Othman, M. F. (2008). A preemptive utility accrual scheduling algorithm for adaptive real time system. *IJCSNS International Journal of Computer Science and Network Security*, 8(5), 57-61.

[7] Moschakis, I. A. & Karatza, H. D. (2012). Evaluation of gang scheduling performance and cost in a cloud computing system. *The Journal of Supercomputing*,

February, 59(2), 975-992.

[8] Hwang, J. & Wood, T. (2012). *Adaptive Dynamic Priority Scheduling for Virtual Desktop Infrastructures*. Proceedings of the 2012 IEEE 20th International Workshop on Quality of Service.

[9] Jann, J., Pattnaik, P., Franke, H., Wang, F., Skovira, J. & Riordan, J. (1997). *Modeling of Workload in MPPs*. Proceedings of Workshop Job Scheduling Strategies for Parallel Processing (pp. 95-116).

[10] Kargahi, M. & Movaghar, A. (2006). A method for performance analysis of earliest-deadline-first scheduling policy. *The Journal of Supercomputing*, 37(2), 197-222.

[11] Paul, M., Samant, D. & Sanyal, G. (2011). Dynamic job scheduling in cloud computing based on horizontal load balancing. *International Journal of Computer Technology and Application*, 2(5), 1552-1556.

[12] Jettee, M. & Feitelson, D. (1997). *Improved Utilization and Responsiveness with Gang Scheduling*. Proceedings of Workshop Job Scheduling Strategies for Parallel Processing (pp. 238-261).

[13] Fujimoto, R., Malik, A. & Park, A. (2010). Parallel and distributed simulation in the cloud. *International Simulation Magazine, Society for Modeling and Simulation*, 1(3).

[14] Fujimoto, R. (1999). *Parallel and Distributed Simulation*. Proceedings of 31st Conference Winter Simulation: Simulation-A Bridge to the Future (1, pp. 122-131)

[15] Fujimoto, R., Malik, A. & Park, A. (2009). *Optimistic Synchronization of Parallel Simulations in Cloud Computing Environments*. Proceedings of IEEE International Conference on Cloud Computing (pp. 49-56).

[16] Tayal, S. (2011). Tasks scheduling optimization for the cloud computing systems. *International Journal of Advanced Engineering Sciences and Technologies*, 5(2), 111-115.

[17] Das, S., Viswanathan, H. & Rittenhouse, G. (2003). *Dynamic Load Balancing Through Coordinated Scheduling in Packet Data Systems*. 22nd Annual Joint Conference of the IEEE Computer and Communications (pp. 786-796)

[18] Etsion, Y. & Tsafrir, D. (2005). *A Short Survey of Commercial Cluster Batch Schedulers*. Technical Report 2005-13. The Hebrew University of Jerusalem.

[19] Lin, Y. (1992). *Parallelism Analyzers for Parallel Discrete Event Simulation*. ACM Transactions on Modeling and Computer Simulation, 2(3), 239-264.

[20] Wiseman, Y. & Feitelson, D. (2003). *Paired Gang Scheduling*. IEEE Transactions on Parallel and Distributed Systems, 14(6), 581-592.

[21] Zhang, Y., Franke, H., Moreira, J. & Sivasubramaniam, A. (2003). *An Integrated Approach to Parallel Scheduling Using Gang-Scheduling, Backfilling, and Migration*. IEEE Transactions on Parallel and Distributed Systems, 14(3), 236-247.