Distributed Loop Scheduling Schemes for Cloud Systems

Yiming Han and Anthony T. Chronopoulos

Department of Computer Science
University of Texas at San Antonio
San Antonio, TX, USA
Email: {yhan, atc}@cs.utsa.edu

***Abstract*—Cloud computing infrastructure offers the computing resources as a homogeneous collection of virtual machine instances by different hardware configurations, which is transparent to end users. In fact, the computational powers of these virtual machine instances are different and behaves as a heterogeneous environment. Thus, scheduling and load balancing for high performance computations become challenging on such systems. In this paper, we propose a hierarchical distributed scheduling scheme suitable for parallel loops with independent iterations on a cloud computing system. We also evaluate various performance aspects associated with our distributed scheduling scheme.**

***Keywords*-Cloud computing, Heterogeneous, Scheduling, Load balance.**

I. Introduction

High performance scientific computing problems have been solved using cluster computing, or computational Grids in the past. Now, Cloud Computing is emerging as the next generation of large-scale scientific computing platforms, eliminating the need of expensive computing hardware and providing user friendly concurrent programming environments. Scientific applications usually contain large loops, which are the largest source of parallelism for high performance computing research. Some independent loop iterations can be divided and allocated to different processors or computers to reduce the total execution time on shared memory or distributed memory computing systems. Previous research [1] [2] [3] [4] proposed some loop scheduling schemes to assign various chunks to each processor. And loop scheduling can be categorized into static and dynamic. Static scheduling schemes determine the task allocation to the processors prior to the execution of the application. Dynamic scheduling (or self-scheduling) is an automatic loop scheduling method in which idle processors request new loop iterations to be assigned to them during run time (the execution of the application). An algorithm for the Feedback-Guided Dynamic Loop Scheduling (FGDLS) to schedule a nested serial/parallel loop is proposed in [5]. An adaptive chunk self-scheduling scheme is proposed (in [6]) to reduce the scheduling overhead. In [7] [8], the authors have proposed new improved self-scheduling schemes named NGSS and ANGSS. A two-phase scheme is proposed to solve parallel regular loop scheduling problem in heterogeneous grid computing environments in [9]. A version of the FSS algorithm is proposed and considered for implementation in virtual machine scheduling in cross-cloud environment [10]. Some results [11] also focus on loops with dependencies. Recent research results [12] [13] have been reported for designing loop self-scheduling methods for grid. In [14] [15] [16], the heterogeneity of different cluster systems was considered, in order to get better load balancing.

Presently, cloud computing platforms are growing in popularity. It provides scalable, flexible, reliable and on-demand computing and storage resources over a network. There are some commercial cloud providers, such as Amazon EC2, Microsoft Azure, Salesforce Service Cloud and Google Cloud. Some open source cloud projects for research and development also exist, for example, OpenStack, Eucalyptus, CloudStack and Ganeti. The cloud systems are introduced in [17] and references there in. There are also much research for cloud systems. In [18], a provisioning technique that automatically adapts to workload changes related to applications with Quality of Services (QoS) in large, autonomous, and highly dynamic environments is proposed. [19] extends Grid workflow middleware to compute clouds in order to speed up executions of scientific workflows. Energy consumption of large scale data centers cloud systems has become a prominent problem and received much attention. A hierarchical scheduling algorithm for applications, to minimize the energy consumption of both servers and network devices is proposed in [20]. The problem of provisioning physical servers to a sequence of jobs, and reducing the total energy consumption is studied in [21]. In [22], a Master-Worker model is used for a case study of an application of a parallel simulation optimization deployed on a private Cloud. [23] reported that the effect of some critical parameters (allocation percentages, real-time scheduling decisions and co-placement) on the performance of virtual machines. Cloud computing can be used for solving some computational intensive jobs in high performance computing research area. Clouds are becoming an alternative to clusters, grids, and parallel production environments for scientific computing applications. However, virtualization

and resource time-sharing may introduce performance overheads for the demanding scientific computing workloads. The performance of cloud computing services for scientific computing workloads is studied in [24].

Cloud computing platforms provide computing service to users by virtualization technology [25]. For high performance computing applications, we can use cloud to virtualize clusters on cloud systems. These virtual machines can share the same physical hardware or different physical hardware with various system load and user load and cloud system use a fair-share balancing algorithm that gives equal time to each virtual machine. However, because of limited resources, the virtualized cluster is not private and the resources are shared by many users, which means the virtualized cluster may act as a heterogeneous computing environment at running time. Thus, the heterogeneity should be taken into account to improve resource utilization and reduce load imbalance. MapReduce [26] is a general concurrent programming framework for scheduling job-tasks on cloud systems. However, MPI concurrent programming yields higher performance than MapReduce. Previous research [27] [28] [29] reported that the performance on virtual machines is lower than the physical system. They analyzed message passing (MPI) parallel applications on different cloud systems and reported that communication overhead is a substantial slowdown factor for cloud systems. In this paper, we design a distributed Master-Worker model for self-scheduling schemes on cloud. We implement these schemes on the FlexCloud system. Our experiments demonstrate the good load balance and good performance of the proposed schemes.

The rest of the paper is organized as follows. In Section 2, we review simple loop self-scheduling schemes. In Section 3, we describe the distributed schemes. In Section 4, experiments and results are presented. In Section 5, conclusions are drawn.

II. LOOP SELF-SCHEDULING SCHEMES

In this section, we review some previously proposed dynamic loop scheduling schemes. These loop scheduling schemes were implemented using a Master-Worker architecture model

A. Notations:

The following are common notations used throughout the whole paper:

- I is the total number of iterations or tasks of a parallel loop;
- p is the number of workers (i.e. processors) in the parallel or heterogeneous system which execute the computational tasks;
- P_1, P_2, ..., P_p represent the p workers in the system;
- A few consecutive iterations are called a *chunk*. C_i is the chunk-size at the i-th scheduling step (where: $i = 1, 2, ...$);
- N is the number of scheduling steps;
- t_j, $j = 1, .., p$, is the execution time of P_j to complete all its tasks assigned to it by the scheduling scheme;
- $T_p = \max_{j=1,..,p}(t_j)$, is the parallel execution time of the loop on all p workers;

In a generic self-scheduling scheme, at the i-th scheduling step, the master computes the chunk-size C_i and the remaining number of tasks R_i:

$$R_0 = I, \qquad C_i = f(R_{i-1}, p), \qquad R_i = R_{i-1} - C_i \quad (1)$$

where $f(.,.)$ is a function possibly of more inputs than just R_{i-1} and p. Then the master assigns to a worker processor C_i tasks. Imbalance depends on the execution time gap between t_j, for $j = 1, \ldots, p$. This gap may be large if the first chunk is too large or (more often) if the last chunk (called the *critical chunk*) is too small.

The different ways to compute C_i have given rise to different scheduling schemes. Some widely used examples are the following. These methods are studied or extended in [1], [30] and references therein.

Trapezoid Self-Scheduling (TSS) $C_i = C_{i-1} - D$, with (chunk) decrement : $D = \left\lfloor \frac{(F-L)}{(N-1)} \right\rfloor$, where: the first and last chunk-sizes (F,L) are user/compiler-input or $F = \left\lfloor \frac{I}{2p} \right\rfloor$, $L = 1$. The number of scheduling steps assigned: $N = \left\lceil \frac{2*I}{(F+L)} \right\rceil$. Note that $C_N = F - (N-1)D$ and $C_N \geq 1$ due to integer divisions.

Factoring Self-Scheduling (FSS) $C_i = \lceil R_{i-1}/(\alpha p) \rceil$, where the parameter α is computed (by a probability distribution) or is suboptimally chosen $\alpha = 2$. The chunk-size is kept the same in each *stage* or *round* (in which all processors are assigned a chunk of the same size) before moving to the next stage. Thus $R_i = R_{i-1} - pC_i$ (where $R_0 = I$) after each stage.

Guided Self-Scheduling (GSS) $C_i = \lceil R_{i-1}/p \rceil$. In the last steps too many small chunks are assigned. It assigns large chunks initially, which implies reduced communication/scheduling overheads only in the beginning. A modified version $GSS(k)$ with minimum assigned chunk-size k (chosen by the user) attempts to improve on the weaknesses of GSS.

Remark 1: The schemes described above have been called Simple Loop Self-Scheduling Schemes. In the distributed versions of these schemes (called: DTSS, DFSS, DGSS) the master takes into account the computing rates of the workers in determining the sizes of the tasks to be assigned [**?**].

III. DISTRIBUTED LOOP SCHEDULING SCHEMES

In this section, we review the methodology for distributed loop scheduling schemes. Distributed schemes (DTSS, DFSS, DGSS) are derived from the corresponding simple

loop scheduling schemes by considering different computing powers of workers.

A. Terminology:

- V_j = $Speed(P_j)/min_{1\leq i\leq p}\{Speed(P_i)\}$, $j = 1, ..., p$, is the virtual power of P_j (computed by the master), where Speed(P_j) is the processing speed of P_j. That is a standardized computing power in the current cluster.
- $V = \sum_{j=1}^{p} V_j$ is the total virtual computing power of the cluster.
- DC is the distributed chunk size for one worker request, in a single scheduling step of distributed self-scheduling scheme.

Master:

- **(1)** Compute V_j for each worker
  (a) Receive $Speed(P_j)$;
  (b) Compute all V_j;
  (c) Send all V_j ;
- **(2)** Assign work and get the results
  (a) While there are unassigned tasks, if a request arrives, put it in the Request Queue.
  (b) Pick a request from the queue and get its virtual power V_j. If there are computed results in this request, Result Collector receives them first. Then Task Scheduler compute the next chunk size DC to assign. The followings are DTSS, DFSS and DGSS algorithms to compute the next chunk DC:

DTSS:
$Current$ is chunk size in the current step of TSS.
Initialization: $F = \lfloor \frac{I}{2V} \rfloor$, $L = 1, N = \left\lceil \frac{2*I}{(F+L)} \right\rceil$,
$D = \left\lfloor \frac{(F-L)}{(N-1)} \right\rfloor, Current = F$

DFSS:
DC_{sum} is the assigned work in current stage.
Initialization: $R = I, \alpha = 2.0, DC_{sum} = 0$

DGSS:
Initialization: $R = I$

Algorithm 1 Calculate DC, DTSS

$DC = 0$;
for $k = 1 \rightarrow V_j$ **do**
  $DC = DC + Current$;
  $Current = Current - D$;
end for
return DC;

Algorithm 2 Calculate DC, DFSS

$DC = \lceil R/(\alpha V) \rceil * V_j$;
$DC_{sum} = DC_{sum} + DC$;
if (Master has assigned all the work in the current stage) **then**
  { Goto next stage and update the remaining work. }
  $R = R - DC_{sum}$;
  $DC_{sum} = 0$;
end if
return DC;

Algorithm 3 Calculate DC, DGSS

$DC = \lceil R/(A) \rceil * V_j$;
$R = R - DC$;
return DC;

Worker :

- **(1)** Send $Speed(P_j)$;
- **(2)** Send a request;
- **(3)** Wait for a reply;
  IF (There is unassigned work)
  {
    Compute the new work;
    Return the results and send another request;
    Go back to **(2)**;
  }
  ELSE
    Terminate;

IV. Implementation and Results

A. Applications

- Mandelbrot Set

  The Mandelbrot Set is a doubly nested loop without dependencies. The computation of one column of the Mandelbrot matrix is considered the smallest schedulable unit. The following loops are used for computing the Mandelbrot Set.

```
MSetLSM(MSet,nx,ny,xmin,xmax,ymin,ymax,maxiter)
BEGIN
 FOR iy = 0 TO ny-1 DO
  cy = ymin+iy*(ymax - ymin)/(ny - 1)
  FOR ix = 0 TO nx-1 DO
   cx = xmin+ix*(xmax - xmin)/(nx - 1)
   MSet[ix][iy]=MSetLevel(cx,cy,maxiter)
  END FOR
 END FOR
END
MSetLevel(cx,cy,maxiter)
BEGIN
 x = y = x2 = y2 = 0.0, iter = 0
```

```
WHILE(iter<maxiter)AND(x2+y2<2.0)DO
  temp = x2 - y2 + cx
  y = 2*x*y + cy
  x = temp
  x2 = x*x
  y2 = y*y
  iter = iter + 1
END WHILE
RETURN(iter)
END
```

- Adjoint Convolution

  This application is decreasing load imbalance, some iterations at the beginning take most of running time, the i_{th} iteration's running time is $O(N^2 - i)$.

```
BEGIN
  FOR I = 1 TO N * N DO
    FOR J = I TO N * N DO
      A(I) = A(I) + X * B(J) * C(J - I)
    END FOR
  END FOR
END
```

B. Cloud Platform

We use FlexCloud of Institute for Cyber Security(ICS) at University of Texas at San Antonio. The ICS FlexCloud is one of the first dedicated Cloud Computing academic research environments. It offers significant capacity and similar design features found in Cloud Computing providers, including robust compute capability and elastic infrastructure design. FlexCloud highlights currently include:

- 5 Racks of Dell R410, R610, R710, and R910s consisting of 748 processing cores, 3.44TB of RAM, and 144TB of total storage.
- Redundant 10GB network connectivity provides high performance access between all nodes.
- Powered by Joyent SmartDataCenter [31] providing the highest performance virtualization and analytics. And Joyent SmartOS provides a combination of hardware and operating system virtualization to support efficient, reliable and high performing cloud computing.
  - Joyent uses the HPC model of management: one headnode PXE boots compute nodes
  - SmartOS is a RAM disk based image (nothing stored on disk) which makes it very easy to upgrade headnodes/computenodes
  - SmartOS uses the disks on the local nodes to back the SmartMachines and Virtual Machines using ZFS
  - SmartOS has DTrace which allows for monitoring all VMs with little overhead for maximum observability
  - SmartOS has the best multitenant defenses to prevent one tenant from affecting others on the box

C. Results

In this section, we compare the performance of simple scheme and distributed schemes with 8 and 16 workers (processors or virtual machines). The Mandelbrot Set computation domain is [-2.0, 2.0] $\times$ [-2.0, 2.0] and its size is 10K $\times$ 10K and 20K $\times$ 20K. The Adjoint Convolution has a size of $\sqrt{9K} \times \sqrt{9K}$ and $\sqrt{16K} \times \sqrt{16K}$. And the arrays are generated randomly.

For all experiments, we initialize 19 virtual machine instances with one virtual core instance, 1GB memory and 10GB storage. Each instance is loaded with an Ubuntu Linux 10.04 image. GCC/G++ and OpenMP are installed. Because the cloud's utilization is low and the virtual machine instances' computation powers may be the same, we use Stress [32], a simple workload generator, to add various system/user load to the instances. That can help get a more realistic heterogeneous computing environment. The computing power of each virtual machine instance is measured using run time tests. The computing power ratio of virtual machine instances is 8, 8, 8, 5, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1. Each worker can get work proportional to its computing power. If all the virtual machine instances are treated the same, which is a homogeneous computing environment, there is no need to measure the computing power before running and all the ratios are 1. In that case all workers get the same amount of work.

The following loop scheduling schemes are implemented. Simple schemes: TSS, FSS, GSS; distributed schemes: DTSS, DFSS, DGSS. All the schemes are implemented by C++ and MPI.

We note that the parallel time $T_p = max\{T_{comp_1}, T_{comp_2}, \ldots, T_{comp_p}\}$ + Time (for communication and scheduling overhead), where T_{comp_i} is the i_{th} worker total computation time. The *maximum-times-difference* in the workers computation times is a measure of the imbalance of the workers computational load. **(I)** Figure 1 and Figure 2 present the *maximum-times-difference* for Mandelbrot Set with sizes of 10K $\times$ 10K and 20K $\times$ 20K using 8 and 16 workers. And **(II)** Figure 3 and Figure 4 are for Adjoint Convolution with sizes of $\sqrt{9K} \times \sqrt{9K}$ and $\sqrt{16K} \times \sqrt{16K}$. It can be observed that the differences in the case of simple schemes are quite substantial and hence there is considerable load imbalance among the workers. The differences in the case of distributed schemes are much smaller.

From Figures 5 – 8, it can be observed that the execution time of distributed schemes are less than simple schemes. Because distributed schemes use the computing powers for each virtual machine instance before computing the work chunks. Thus, more work is assigned to faster workers which leads to balanced workloads and improved

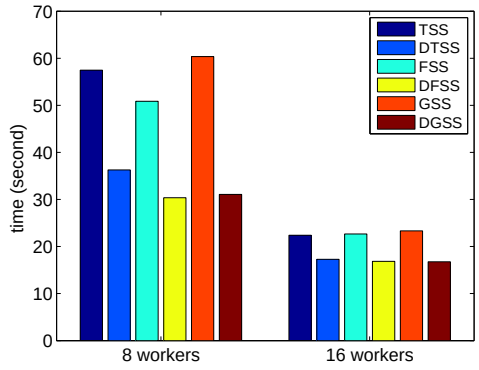

Figure 1. *maximum-times-difference* for Mandelbrot Set with 10K × 10K size between simple and distributed schemes

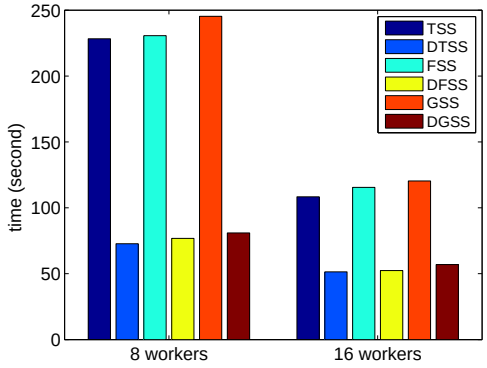

Figure 2. *maximum-times-difference* for Mandelbrot Set with 20K × 20K size between simple and distributed schemes

resource utilization. Figures 9 – 12 present the speedup comparison between simple loop scheduling schemes and distributed loop scheduling schemes with 8 and 16 workers. The speedup is computed by $S_p = \frac{\hat{T}_1}{T_p}$, $\hat{T}_1$ is the execution time for the simple TSS scheme with 8 workers. It can be observed that, as the number of workers increases, the speedup of distributed schemes improves which shows that the schemes are scalable. All the speedups of distributed schemes are better than simple distributed schemes' with 8 and 16 workers.

V. Conclusions and Future work

In this paper, we studied and implemented (in MPI) distributed loop scheduling schemes. The distributed schemes for loop self-scheduling schemes on a cloud system present better performance, especially due to better properties. In the future, we will implement a hierarchical distributed loop scheduling schemes on cloud systems to test the scalability and efficiency.

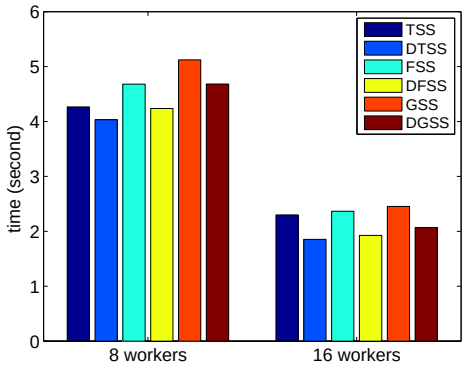

Figure 3. *maximum-times-difference* for Adjoint Convolution with $\sqrt{9K} \times \sqrt{9K}$ size between simple and distributed schemes

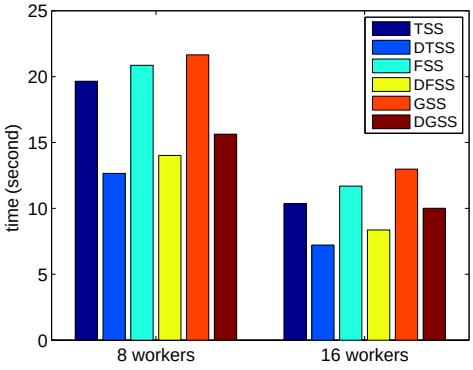

Figure 4. *maximum-times-difference* for Adjoint Convolution with $\sqrt{16K} \times \sqrt{16K}$ size between simple and distributed schemes

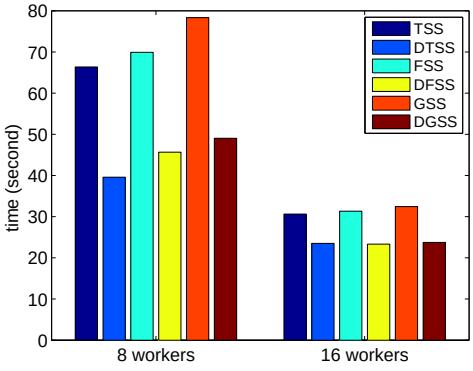

Figure 5. Performance for Mandelbrot Set with 10K × 10K size between simple and distributed schemes

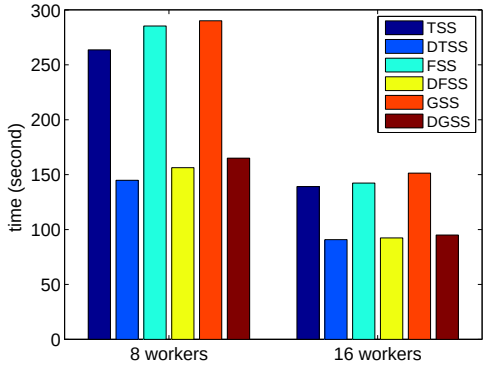

Figure 6. Performance for Mandelbrot Set with 20K $\times$ 20K size between simple and distributed schemes

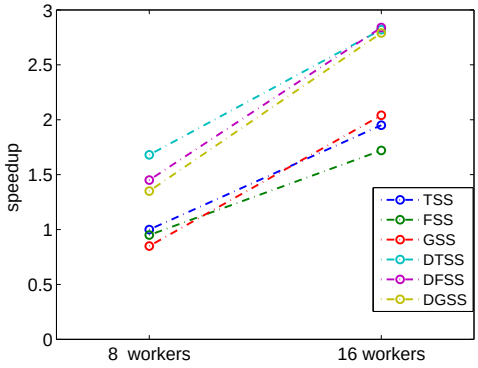

Figure 9. Speedup for Mandelbrot Set with 10K $\times$ 10K size between simple and distributed schemes

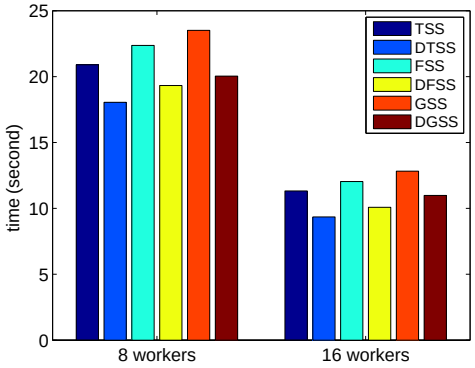

Figure 7. Performance for Adjoint Convolution with $\sqrt{9K} \times \sqrt{9K}$ size between simple and distributed schemes

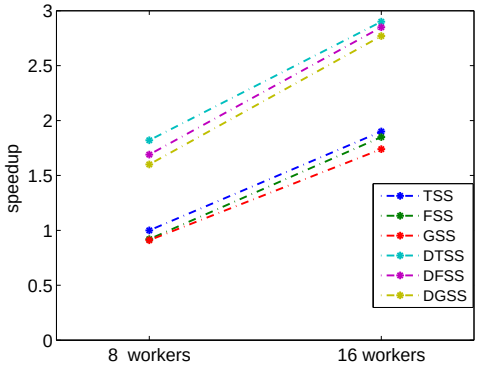

Figure 10. Speedup for Mandelbrot Set with 20K $\times$ 20K size between simple and distributed schemes

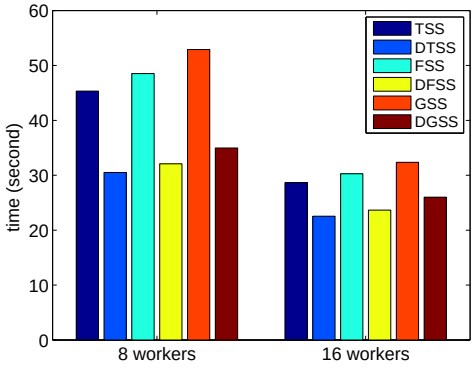

Figure 8. Performance for Adjoint Convolution with $\sqrt{16K} \times \sqrt{16K}$ size between simple and distributed schemes

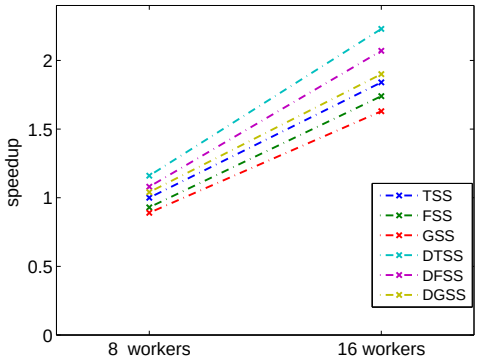

Figure 11. Speedup for Adjoint Convolution with $\sqrt{9K} \times \sqrt{9K}$ size between simple and distributed schemes

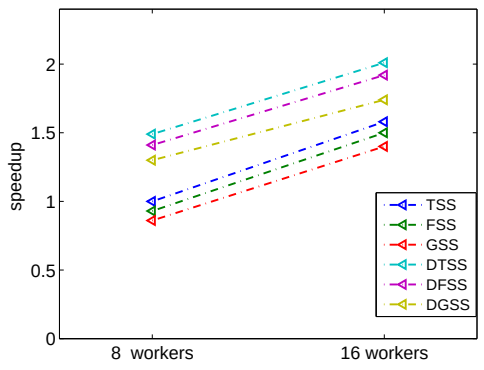

Figure 12. Speedup for Adjoint Convolution with $\sqrt{16K} \times \sqrt{16K}$ size between simple and distributed schemes

ACKNOWLEDGMENT

We gratefully acknowledge the following: (i) the reviewers for their helpful and constructive suggestions, which considerably improved the quality of the manuscript; (ii) support by NSF grant (HRD-0932339) to the University of Texas at San Antonio; and (iii) time grants to access the facilities ofInstitute for Cyber Security(ICS) of University of Texas at San Antonio and FutureGrid at Indiana University, Bloomington.

REFERENCES

[1] E. P. Markatos and T. J. LeBlanc, "Using processor affinity in loop scheduling on shared-memory multiprocessors," *IEEE Trans. on Parallel and Distributed Systems*, vol. 5, pp. 379–400, 1994.

[2] S. Chen and J. Xue, "Partitioning and scheduling loops on NOWs," *Computer Communications*, vol. 22, 1999.

[3] A. Kejariwal, A. Nicolau, and C. Polychronopoulos, "History-aware self-scheduling," *International Conference on Parallel Processing Parallel Processing(ICPP)*, pp. 185–192, 2006.

[4] T. H. Kim and J. M. Purtilo, "Load balancing for parallel loops in workstation clusters," *International Conference on Parallel Processing*, pp. 182–190, 1996.

[5] T. Tabirca, S. Tabirca, and L. Yang, "An O(log p) algorithm for the discrete feedback guided dynamic loop scheduling," *20th International Conference on Advanced Information Networking and Applications(AINA)*, p. 6 pp., 2006.

[6] P. Li, Q. Ji, Y. Zhang, and Q. Zhu, "An adaptive chunk self-scheduling scheme on service grid," *IEEE Asia-Pacific Services Computing Conference(APSCC)*, pp. 39–44, 2008.

[7] J. Hai, G. GuangR, X. Zhiwei, and C. Hao, "An efficient parallel loop self-scheduling on grid environments," *Network and Parallel Computing*, pp. 92–100, 2004.

[8] C.-T. Yang and S.-C. Chang, "A parallel loop self-scheduling on extremely heterogeneous pc clusters," *Proc. of Intl Conf. on Computational Science*, pp. 1079–1088, 2003.

[9] K.-W. Cheng, C.-T. Yang, C.-L. Lai, and S.-C. Chang, "A parallel loop self-scheduling on grid computing environments," *Proceedings of International Symposium on Parallel Architectures, Algorithms and IEEE Networks*, pp. 409–414, 2004.

[10] D. Theng and K. Hande, "VM management for cross-cloud computing environment," *Proceedings of International Conference on Communication Systems and Network Technologies(CSNT)*, pp. 731–735, 2012.

[11] F. M. Ciorba, T. Andronikos, I. Riakiotakis, A. T. Chronopoulos, and G. Papakonstantinou, "Dynamic multi phase scheduling for heterogeneous clusters," *Proc. of the 20th IEEE Int. Parallel Distributed Processing Symposium(IPDPS)*, p. 10 pp., 2006.

[12] P. Li, Q. Ji, Y. Zhang, and Q. Zhu, "An adaptive chunk self-scheduling scheme on service grid," *IEEE Asia-Pacific Services Computing Conference(APSCC)*, pp. 39–44, 2008.

[13] C.-C. Wu, C.-T. Yang, K.-C. Lai, and P.-H. Chiu, "Designing parallel loop self-scheduling schemes using the hybrid MPI and openMP programming model for multi-core grid systems," *The Journal of Supercomputing*, vol. 59, pp. 42–60, 2012.

[14] Y. He, J. Liu, and H. Sun, "Scheduling functionally heterogeneous systems with utilization balancing," *2011 IEEE International Parallel Distributed Processing Symposium (IPDPS)*, pp. 1187–1198, 2011.

[15] S. Penmatsa, A. Chronopoulos, N. Karonis, and B. Toonen, "Implementation of distributed loop scheduling schemes on the teragrid," *IEEE International Parallel and Distributed Processing Symposium(IPDPS)*, pp. 1–8, 2007.

[16] A. T. Chronopoulos, S. Penmatsa, J. Xu, and S. Ali, "Distributed loop-scheduling schemes for heterogeneous computer systems," *Concurrency and Computation:Practice and Experience*, pp. 771–785, 2006.

[17] K. Hwang, J. Dongarra, and G. C. Fox, *Distributed and Cloud Computing: From Parallel Processing to the Internet of Things*. Morgan Kaufmann, 2011.

[18] R. Calheiros, R. Ranjan, and R. Buyya, "Virtual machine provisioning based on analytical performance and Qos in cloud computing environments," *2011 International Conference on Parallel Processing(ICPP)*, pp. 295–304, 2011.

[19] S. Ostermann, R. Prodan, and T. Fahringer, "Extending grids with cloud resource management for scientific computing," *2009 10th IEEE/ACM International Conference on Grid Computing*, pp. 42–49, 2009.

[20] G. Wen, J. Hong, C. Xu, P. Balaji, S. Feng, and P. Jiang, "Energy-aware hierarchical scheduling of applications in large scale data centers," *2011 International Conference on Cloud and Service Computing (CSC)*, pp. 158–165, 2011.

[21] Y.-C. Hsu, P. Liu, and J.-J. Wu, "Job sequence scheduling for cloud computing," *Proceedings of the 2011 International Conference on Cloud and Service Computing*, pp. 212–219, 2011.

[22] G. V. Mc Evoy, B. Schulze, and E. L. M. Garcia, "Performance and deployment evaluation of a parallel application on a private cloud," *Concurrency and Computation:Practice and Experience*, pp. 2048–2062, 2011.

[23] "The effects of scheduling, workload type and consolidation scenarios on virtual machine performance and their prediction through optimized artificial neural networks," *Journal of Systems and Software*, pp. 1270–1291, 2011.

[24] A. Iosup, S. Ostermann, M. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema, "Performance analysis of cloud computing services for many-tasks scientific computing," *IEEE Transactions on Parallel and Distributed Systems*, pp. 931–945, 2011.

[25] J. E. Simons and J. Buell, "Virtualizing high performance computing," *ACM SIGOPS Operating Systems Review*, pp. 136–145, 2010.

[26] W.-C. Shih, S.-S. Tseng, and C.-T. Yang, "Performance study of parallel programming on cloud computing environments using MapReduce," *2010 International Conference on Information Science and Applications(ICISA)*, pp. 1–8, 2010.

[27] J. Ekanayake and G. Fox, "High performance parallel computing with clouds and cloud technologies," *Proceedings of the first International Conference on Cloud Computing*, pp. 20–38, 2010.

[28] C. Evangelinos and C. N. Hill, "Cloud Computing for parallel Scientific HPC Applications: Feasibility of Running Coupled Atmosphere-Ocean Climate Models on Amazon's EC2," *Computability and Complexity in Analysis*, pp. 159–168, 2008.

[29] A. Ranadive, M. Kesavan, A. Gavrilovska, and K. Schwan, "Performance implications of virtualizing multicore cluster machines," *Proceedings of the 2nd workshop on System-level virtualization for high performance computing*, pp. 1–8, 2008.

[30] I. Banicescu, V. Velusamy, and J. Devaprasad, "On the scalability of dynamic scheduling scientific applications with adaptive weighted factoring," *Cluster Computing*, pp. 215–226, 2003.

[31] Joyent, http://joyent.com/.

[32] Stress, http://weather.ou.edu/~apw/projects/stress/.