

Using Swarm Intelligence to Optimize the Energy Consumption for Distributed Systems

Neil Bergmann¹, Yuk Ying Chung², Xiangrui Yang², Zhe Chen², Wei-Chang Yeh³, Xiangjian He³ & Raja Jurdak⁴

¹ School of Information Technology and Electrical Engineering, The University of Queensland, Australia

² School of Information Technologies, University of Sydney, Australia

³ University of Technology Sydney, Australia

⁴ CSIRO ICT Centre, Pullenvale, Australia

Correspondence: Yuk Ying Chung, School of Information Technologies, University of Sydney, NSW 2006, Australia. Tel: 61-2-9036-9109. E-mail: ychung@it.usyd.edu.au

Received: November 16, 2012

Accepted: April 19, 2013

Online Published: May 21, 2013

doi:10.5539/mas.v7n6p59

URL: <http://dx.doi.org/10.5539/mas.v7n6p59>

Abstract

Large, distributed, network-based computing systems (also known as Cloud Computing) have recently gained significant interest. We expect significantly more applications or web services will be relying on network-based servers, therefore reducing the energy consumption of these systems would be beneficial for companies to save their budgets on running their machines as well as cooling down their infrastructures. Dynamic Voltage Scaling can save significant energy for these systems, but it faces the challenge of efficient and balanced parallelization of tasks in order to maximize energy savings while maintaining desired performance levels. This paper proposes our Simplified Swarm Optimization (SSO) method to reduce the energy consumption for distributed systems with Dynamic Voltage Scaling. The results of SSO have been compared to the most popular evolutionary Particle Swarm Optimization (PSO) algorithm and have shown to be more efficient and effective, reducing both the execution time for scheduling and makespan.

Keywords: energy optimization, evolutionary algorithm, distributed computing

1. Introduction

According to recent research on energy consumption, the electricity usage on servers in U.S. in 2005 represents 0.6% of the total electricity consumption of the whole country, and the number goes to 1.2% when the cooling infrastructures are also included (Koomey, 2008). In the past few years, out of the interest of large scalability, cost efficiency and performance, the concept of cloud computing has become very popular among companies that require huge amounts of computation, and nowadays the rapid growth of both experimental and commercial cloud services has brought a significant influence on building large data centers, web services, and the whole Internet, together with a tremendous growth of energy consumption on these machines.

Dynamic Voltage Scaling (Lee & Zomaya, 2009) is a power management technique that can optimize the energy efficiency on distributed systems with or without influencing the overall performance depending on our goals. In order to perform dynamic voltage scaling effectively, we need to find out the critical path in the task graph. Unfortunately, the problem of finding out the critical path itself is NP-Complete (Garey & Johnson, 1990), which means that the solutions can be hard to compute in a reasonable amount of time. In this paper, we propose Simplified Swarm Optimization (SSO) to achieve better performance. The results have been compared with the most popular evolutionary algorithm Particle Swarm Optimization (PSO) (Kennedy & Eberhart, 1995). We explain the design of our experiments, how the algorithms work and why would they be effective in solving distributed computing problems.

To evaluate the performance, we have focused on three parameters: makespan, time and energy consumption. The evaluation approach will be introduced in Sections 2 and 3, and the experimental results and analysis will be presented in Section 4.

2. Experiment Design

2.1 Overview

A distributed computing task can be presented as a directed graph with weights on each node indicating the amount of computation that is needed to complete each task respectively.

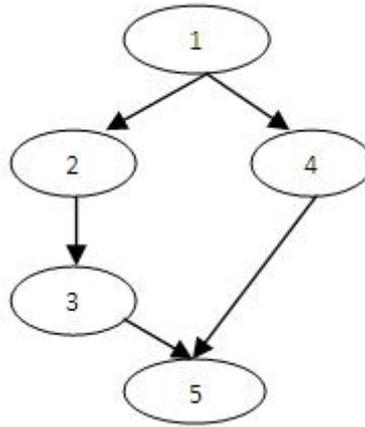


Figure 1. An example graph

Figure 1 is an example graph, with the nodes 1 to 5 indicating 5 different individual tasks that are needed for the whole computation. With dependencies to each other (shown as arrows), we know that in order to start calculating task 5, task 3 and task 4 need to be finished beforehand. Inside a distributed system, a task scheduler needs to distribute the tasks on different machines to make the overall performance efficient. For the example that is given in Figure 1, a possible solution is to utilize 2 threads: one of them is made of tasks 1, 2, 3, 5 and the other one is made of tasks 1, 4 and 5.

Parallel and distributed computing does not always provide a better performance compared to sequential computing, with the delay and overhead caused by message sending and unbalanced parallelism (http://en.wikipedia.org/wiki/Distributed_computing; http://en.wikipedia.org/wiki/Parallel_computing). Utilizing dynamic voltage scaling can help us to optimize the time and energy consumption for unbalanced tasks in a distributed system.

2.2 Dynamic Voltage Scaling

Dynamic Voltage Scaling (DVS) is a power management function that allows software (e.g. port through BIOS) to scale up or scale down the voltage supply on components (such as RAM, CPU and disk) inside a computing system. The power consumption (Rabaey, 1996) P on a device can be presented as follows.

$$P = CV^2f \quad (1)$$

where C is the capacitance being switched per clock cycle; V is the supply voltage and f is the switching frequency.

DVS has proven to be a very promising technique on DVS-enabled devices. To implement DVS efficiently in a distributed computing system, we need to schedule the tasks on different processors so that we can reach the maximum optimization with minimum energy power to finish the specific task. The available voltage levels are normally pre-set according to the hardware standards. In our case, we have the following four levels according to the Intel Pentium 4 Processor:

Table 1. The pre-set voltage level for the simulated hardware (Note 1)

Level	Voltage (V)	Relative Speed (%)	Energy Reduction (%)
0	1.75	100%	0%
1	1.4	80%	20%
2	1.2	60%	22%
3	0.9	40%	33%

Unfortunately, the algorithm of finding the critical path is NP-Complete, which means that the processing time is super-polynomial to the input size. This renders the computation time prohibitive for problems with large excessively large inputs. To address this issue, we have proposed the new evolutionary optimization algorithm called Simplified Swarm Optimization (SSO), which we present in detail in Section 3. We first present the graph-based model of distributed systems and the key performance parameters to provide context on the problem that requires optimization.

2.3 Graph-Based Representation

We use Directed Acyclic Graph (DAG) file format to store the tasks. A DAG file indicates a tree structure of tasks that need to be computed. In our DAG files, we specify the number of processors, the dependencies between nodes, the processing time for each task on each processor, and the communication delay for each dependency.

The proposed optimization algorithm can allocate the processor for each task as well as the voltage supply for each pair of processor and task. The energy consumption will be calculated according to the voltage supply and the relative speed.

2.4 Performance Parameters

The evaluation will be focused on three different aspects: makespan, time, and the energy optimization.

Makespan is the amount of time needed to finish processing all the tasks including the delays for communications created during the distributed processes. The makespan is a key parameter to evaluate the quality of the task schedule. A good schedule of tasks should be relatively more balanced so that we can achieve good performance.

Time is the amount of time needed to schedule the tasks. Greedy algorithms always provide us with a solution in the shortest time, however since our problem is NP-Complete, we need algorithms that make approximations so that we can calculate the solution in a reasonable amount of time.

Energy consumption is the most important measure in our research, which includes energy needed for computing the tasks, as well as the overhead created during distributing the tasks.

Our goal is to measure these three aspects for different optimization algorithms so that we can achieve the best energy savings with the minimum performance cost.

3. Algorithms

This section introduces the class of swarm intelligence algorithms (http://en.wikipedia.org/wiki/Swarm_intelligence). We briefly revisit the popular Particle Swarm Intelligence approach, and then we introduce our Simplified Swarm Intelligence method.

3.1 Introduction to Swarm Intelligence

The inspiration of Swarm Intelligence (SI) (Beni, 1989) comes from nature. SI systems are often made up of a set of robots or agents, who are operating on an n-dimensional space interacting locally with one and another.

The following will give an introduction to the most popular evolutionary algorithm Particle Swarm Optimization (PSO) and the proposed Simplified Swarm Optimization (SSO). Both of them have been applied to the task scheduling problem in this work.

3.2 PSO

The original idea of Particle Swarm Optimization (PSO) was inspired by the movement of bird flocking. The PSO algorithm mimics the behavior of flying birds and their means of information exchange to solve optimization problems. Each potential solution is seen as a particle with a certain velocity, and “flies” through the problem space. Each particle adjusts its flight according to its own flying experience and its companions’ flying experience. The particle swarms find optimal regions of complex search spaces through the interaction of individuals in a population of particles. PSO has been successfully applied to a large number of difficult combinatorial optimization problems; and it often outperforms Genetic Algorithms (Goldberg, 1989).

In PSO, particles represent candidate solutions in a solution space, and the optimal solution is found through moving the particles in the solution space. An individual particle flies through an n-dimensional search space with a velocity that dynamically changes according to its own experience and other particles existing in the same search space. The velocity changes under the following rule:

$$V_{id(t)} = WV_{id(t-1)} + C_1 R_1 (P_{id(t-1)} - X_{id(t-1)}) + C_2 R_2 (P_{gd(t-1)} - X_{id(t-1)}) \quad (2)$$

$$X_{id(t)} = X_{id(t-1)} + V_{id(t-1)} \quad (3)$$

where $d=1, 2, 3, \dots, S$; t is the round of iteration; V_i and X_i are the velocity and position of the i_{th} particle; P_i is the previous local best of particle I and is called $pbest$; P_a is the previous best position for all particles and is called $gbest$; C_1 and C_2 are positive constants; and R_1 and R_2 are random values. All particles will be assessed on their fitness by a function. A standard procedure for PSO can be described by the following:

- 1) Initialize $t = 0$, and S , and set $P = S$
- 2) Evaluate S and P , and define $gbest$ and $pbest$
- 3) While $t < MAX_ITERATION$:
- 4) Update S using Equation (2) and Equation (3)
- 5) Evaluate S
- 6) Update P and redefine $pbest$ and $gbest$
- 7) $t = t+1$
- 8) END While
- 9) Return $gbest$

3.3 SSO

In this paper, we propose Simplified Swarm Optimization (SSO) which is an adaptation of PSO using discrete values and modifying the mutation operation to be determined randomly between 3 user-set limits. SSO uses random populations with the mutation operation randomly changing each dimension of the particle as determined by the aforementioned limits. This allows the user to change between focusing on finding local optima and expanding the search to cover more of the problem space.

SSO is unique and effective due to its simple search methods. Prior to this, particles are mutated randomly with random dimensions being changed in each cycle. Having each dimension change toward a local optimal or global optimal dimension is very well suited to discrete data and distributed computing applications. Combining with the above algorithms gives a wide variety of random vs. controlled mutation and continuous vs. discrete data optimisation techniques.

The main difference of SSO and PSO is that, SSO does not need to use the velocity and the initial weight; instead the update positions of particles are chosen based on the relationship between the values of the new generated random variable and three pre-defined constants C_w , C_D and C_G ranging from (0, 1):

$$X_{id(t)} = \begin{cases} X_{id(t-1)} & \text{if } newRandom \in [0, C_w) \\ P_{id(t-1)} & \text{if } newRandom \in [C_w, C_p) \\ g_{id(t-1)} & \text{if } newRandom \in [C_p, C_g) \\ x & \text{if } newRandom \in [C_g, 1) \end{cases} \quad (4)$$

where X is the position, P is the local best and g is the global best. During the iterations, for each dimension of the particle, a $newRandom$ variable is generated in the range of (0, 1), and the new position will be chosen between previous position, local best, global best and the current location based on which interval the $newRandom$ lies in.

Based on the concept, SSO is more suitable to deal with discrete variables and PSO is more suitable to deal with continuous variables.

In section 4, we will compare the performance of PSO and the proposed SSO in our experimental model and explore the feasibility of using them as a task scheduler.

4. Performance Evaluation

4.1 Benchmark

To evaluate the performance, we need some comparative data. For example, the energy consumption usually

decreases when the execution time takes longer, and our goal is to find an operating point that balances between the performance and energy saving. Therefore, when evaluating the energy consumption, we also need to take in consideration of the speedups.

The makespan will be compared to the sequential runtime (T_o) of the original tasks so that we can see how different the speedups (Amdahl, 1967) are by using different optimization techniques. To calculate the speedups, we have:

$$Speedup = \frac{T_o}{makespan + time} \quad (5)$$

We will also calculate the raw energy consumption (E_o), to evaluate the results. The energy optimization will be $O_{algo} = E_o / E_{algo}$. We have used hundreds of simulated tasks from the DAG files to test and verify our proposed SSO system. The optimization will be mainly measured by the macro-average of O_{algo} .

4.2 Test Data

In this paper, we use a graph generator to generate a large set of DAG files for the testing data. The graph generator can generate directed acyclic graphs with specified number of edges and nodes, and the amount of computation needed for each task (node) and the communication (edge) is randomly selected from a list of commonly used constant values.

Table 2. The testing result for PSO and SSO

		<i>Makespan (ms)</i>	<i>Time (ms)</i>	<i>Energy (J)</i>
None	Overall	10005777	0	30595583
	Avg	100%	N/A	100%
PSO	Overall	3806335	1079	23955588
	Avg	42.64%	N/A	78.24%
SSO	Overall	2476881	855	24435717
	Avg	27.02%	N/A	79.21%

We have tested our model using sufficiently large amount of data in our experiments. For each test case, we have calculated the makespan, time and energy consumption under no optimization and under optimization using traditional PSO and the proposed SSO. To compare the differences, we have used the macro-average of makespan and energy saving to calculate the average performance of the three testing algorithms: None (no optimization), PSO and SSO.

Table 3. The speedup result for using PSO and SSO

	<i>None</i>	<i>PSO</i>	<i>SSO</i>
Speed up	1	2.63	4.03

4.3 Analysis

Table 2 shows the overall statistics of our experiment. From the table, we can see that both of the two algorithms can achieve over 20% of energy savings and PSO can perform slightly (less than 1%) better than SSO. However, SSO can provide scheduled tasks with better quality, and the makespan of SSO solutions are significantly less than the makespan of PSO (from Table 2). The execution time of SSO is also around 10% smaller than PSO. Table 3 has shown the speedups (calculated using Equation (5)) of PSO and SSO. Our results show that SSO is definitely providing a task schedule that consumes much less time than PSO. Also, we have found that both PSO and SSO can work better when dealing with more complex tasks.

In the experiments, we have also tried various tasks of different topological structures, and the tasks are different on many aspects which include: 1) the number of available processors, 2) the dependencies between the tasks, 3) the depth of critical path, 4) the total number of possible paths. From our results, we have found that the structure of the tasks have very little effect on the energy optimization; however the effect on makespan varies a lot.

4.4 Comparison

4.4.1 Makespan

We have also studied the performance of PSO and SSO on individual test cases. In Figure 2, we have used a chart to compare the makespan of using SSO and PSO. In this work, we would like to test how these two algorithms are different in makespan. According to the results in the previous table (Table 2), the overall average of makespan of SSO is only 65% of PSO, and we have found that it also applies to the individual cases from Figure 2.

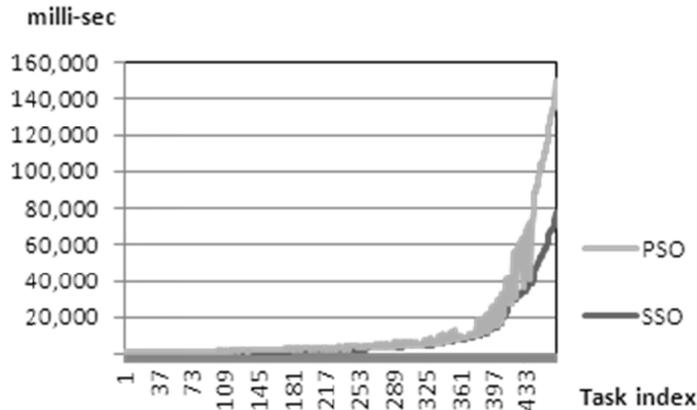


Figure 2. Makespan comparison for SSO and PSO

In Figure 2, the x coordinate indicates the index of test case, and the y coordinate indicates the measurement of makespan. As displayed in the chart, the ratio of PSO makespan and SSO makespan is also relatively stable for most of the cases.

4.4.2 Time

Figure 3 below shows the relationship of the execution time of PSO and SSO. As shown in Figure 3 the x coordinate indicates the test case index (test cases sorted by their execution time), and the y coordinate indicates the time in milli seconds. We can see from Figure 3, the ratio of time between PSO and SSO has shown that SSO is 10% faster than PSO.

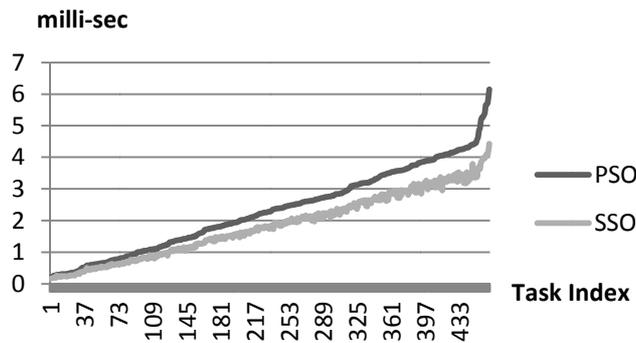


Figure 3. Execution time comparison for PSO and SSO

4.4.3 Energy

The experimental results have shown that both PSO and SSO can deliver over 20% of energy savings. We have studied the performance of PSO and SSO on energy saving. In the graph below, we are showing the O_{PSO} / O_{SSO} for 500 individual test cases.

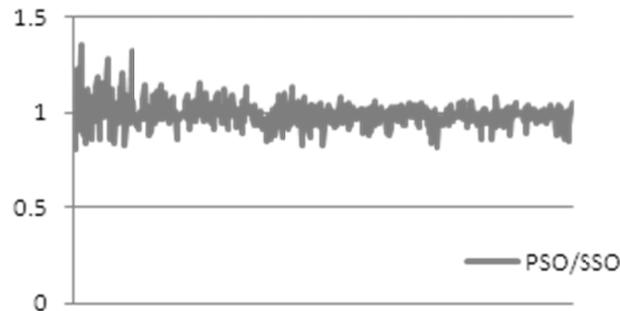


Figure 4. Energy saving comparison for PSO and SSO

Figure 4 shows that the ratio of $O_{\text{PSO}} / O_{\text{SSO}}$ is floating at around 1. Therefore, the results of PSO and SSO are very close to each other.

5. Conclusions and Future Work

Task scheduling is a common problem for distributed systems. In our research, we have improved the computational efficiency by implementing swarm intelligence techniques to solve the task-scheduling problem to reduce the energy cost. Even though the problem itself is NP-Complete, the proposed SSO algorithm can provide a good solution within a reasonable amount of time without delay. In our model, both PSO and SSO algorithms can improve the performance as well as energy consumption, and the amount of energy saved is approximately 21% of the entire energy cost including the pre-calculation and latency when compared with the system without using optimization algorithm.

We have also studied the differences between the proposed SSO and the traditional PSO. Our experimental results have showed that the proposed SSO tends to provide an overall better solution than PSO for energy consumption and task scheduling, though PSO is also acceptable for non time sensitive systems.

Improving the energy efficiency of computation is becoming increasingly more important nowadays with greater reliance on distributed data centers and servers. We have demonstrated that using dynamic voltage scaling and swarm intelligence can improve the energy efficiency as well as the performance. Moreover, both PSO and SSO can be used to schedule tasks for other distributed tasks such as disk drivers, and GPUs.

References

- Amdahl, G. M. (1967). Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities. *Solid-State Circuits Society Newsletter, IEEE*, 12(3), 19-20. <http://dx.doi.org/10.1109/N-SSC.2007.4785615>
- Beni, G. (1989). From Swarm Intelligence to Swarm Robotics. *Swarm Robotics Lecture Notes in Computer Science*, 3342, 1-9. http://dx.doi.org/10.1007/978-3-540-30552-1_1
- Distributed Computing http://en.wikipedia.org/wiki/Distributed_computing
- Garey, M. R., & Johnson, D. S. (1990). *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman Co.
- Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, Mass: Addison-Wesley.
- Kennedy, J., & Eberhart, R. (1995). Full text access may be available. *Particle swarm optimization. IEEE International Conference on Neural Networks*, 4, 1942-1948. <http://dx.doi.org/10.1109/ICNN.1995.488968>
- Kashan, A. H., & Karim, B. (2009). A discrete particle swarm optimization algorithm for scheduling parallel machines. *Computers & Industrial Engineering*, 56(1), 216-223. <http://dx.doi.org/10.1016/j.cie.2008.05.007>
- Koomey, J. G. (2008). Worldwide electricity used in data centers. *Environmental Research Letters*, 3(034008). <http://dx.doi.org/10.1088/1748-9326/3/3/034008>
- Lee, Y. C., & Zomaya, A. Y. (2009). Minimizing energy consumption for precedence-constrained applications using dynamic voltage scaling. *In Proceedings of 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*. <http://dx.doi.org/10.1109/CCGRID.2009.16>
- Parallel Computing http://en.wikipedia.org/wiki/Parallel_computing

Rabaey, J. M. (1996). *Digital Integrated Circuits*. Upper Saddle River, NJ: Prentice Hall.
Swarm Intelligence http://en.wikipedia.org/wiki/Swarm_intelligence

Notes

Note 1. Notice that the energy reduction is under the ideal situation and in real world, we have to consider the communication delay