Scientific Research and Essays Vol. 7(6), pp. 647-659, 16 February, 2012
Available online at http://www.academicjournals.org/SRE
DOI: 10.5897/SRE11.1024
ISSN 1992-2248 ©2012 Academic Journals

Full Length Research Paper

An analysis of super-linear speedup for master-slave model

Wu Weining[1]*, Sihon Crutcher[2] and Xu Runzhang[3]

[1]School of Computer Science and Technology, Harbin Institute of Technology, 150001, People's Republic of China.
[2]US Army RDECOM AMRDEC Weapons Development and Integration Directorate Huntsville, Alabama USA 35898.
[3]College of Science, Harbin Engineering University, 150001, People's Republic of China.

Accepted 10 January, 2012

Recently, large-scale computations have been used in many real application areas. At the same time, there exists more parallel computing technique on cluster in order to meet these demands. In this paper, a speedup analysis of Master-Slave application on heterogeneous cluster is investigated. A task allocation model is set up and its theoretical analysis of execution time is developed. A more accurate upper bound of speedup is derived under some conditions by virtue of the task allocation model. Furthermore, this theoretical analysis is verified by a group of experiments and the experimental results show that the speedup increases nonlinearly and rapidly which is caused by the task optimization in the process.

Key words: Master-Salve, speedup, cluster computing, heterogeneous cluster, super linear speedup.

INTRODUCTION

In the recent years, fast and efficient computers are high demand in many applications such as scientific, engineering, energy resources, medical, military, artificial intelligence and some basic research areas. In these tasks, the common point is that large-scale computation is needed so that parallel processing computers are suitable to meet these demands. Among all the models in parallel computing, Master-Slave model is one of the most popular models. In this model, one or more processes which are so-called Master processes generate work and allocate it to worker processes. With different evaluations of task allocation, it is believed that different computations are derived. This is also the main topic of this present paper. Before we give details of this paper, we like to concern how to evaluate the computing performance first. Speedup is always used to evaluate the performance of a parallel model. Speedup is a measure by capturing the relative benefit of solving a problem in a parallel environment. Although, there exist different definitions, speedup is basically the ratio of sequential and parallel execution times on the same problem. The upper bound of speedup for a fixed size problem is established by Amdahl (1967). Recently, it has been used to evaluate the performance of large-scale data processing on multi-core machine or cluster Dutta et al. (2011). Then, Gustafson (1988) shows that it is possible to increase the size of task in order to avoid the strict limits of Amdahl's law and then the more effective speedup metric can be obtained (Yang et al., 2011). Both Amdahl's and Gutafson's results have a far-reaching impact on parallel computing (Snyder and Sterling, 2000) in the last millennium or grid computing (Gonzalez-Velez and Cole, 2008) recently. There is a rich literature (Head and Govindaraju, 2007, 2009; Almeida et al., 2006; Williams, 2011; Goswami et al., 2010; Bastian et al., 2008; Li et al., 1999; Silva da FAB and Senger, 2011; Shylo et al., 2011) on study of performance of parallel systems.

In this paper, we study the influence of different task allocations to the speedup with the Maser-Slave model on heterogeneous clusters which is first considered by Yero and Henriques (2007). We aim to find the optimal task

*Corresponding author. E-mail: wuweining@yahoo.com.cn. Tel: 86-13936332059.

allocations by advance testing. In Yero and Henriques (2007), they obtained the analytic expression for the execution time by constructing the application, cluster and execution models. They defined speedup and derived speedup bounds based on the inherent parallelism of the application and the aggregated computing power of the cluster. An analytical expression for efficiency is also derived and used to define scalability of the algorithm-cluster combination based on the isoefficiency metric. Furthermore, they established the necessary and sufficient conditions for an algorithm cluster combination to be scalable which are easy to verify and use in practice. But they did not consider the influence of the task allocation to the speedup. We got a more accurate upper bound of the speedup under some conditions for the task allocation model. Then, we gave a process to find the optimal task allocation, so-call advance test. Furthermore, we design an experiment to verify the theoretical analysis. By analyzing the experiment data we find some nonlinear phenomena which show some interesting results of the parallel computation. It is also interesting to find a phenomenon which we define as super-speedup.

MODELS

Definitions and previous work

Here, we are going to recall the heterogeneous clusters model which was first given in Yero and Henriques (2007). The goal of the present paper is not to form a new model, but to find new phenomenon related to this heterogeneous cluster model. In order to save the time of reader, we summarize some definitions and assumptions in here. First, we give some definitions which will be used in this paper:

P: The number of computers forming the cluster;
$C = \{c_1, c_2, ..., c_P\}$: The set of computers forming the cluster. Associated with each computer, there is an attribute t_{op_i}, $1 \le i \le P$, including the time spent by computer i to execute an operation. The value of t_{op_i} is given in $secs/op$. Associated with each link between computer c_1 and c_i, there is an attribute t_{c_i}. The value of t_{c_i} is expressed in $sec/unit$ where $unit$ may be bytes, words or any other data unit. By definition, t_{c_1} is 0. In the application, the computers in the cluster may have twin or more cores which can influence on the computing ability. Here, it can be reflected on t_{op_i}, whose value is smaller than it in a single core computer. Parallel applications are usually based on tasks graphs (Almeida et al., 2006; Goswami et al., 2010; Yan et al., 1996; Sanjay and Vadhiyar, 2008) with the nodes representing the tasks and the links representing data flow. The application model is the base for our study. The marks which we used in the present paper are presented as follows:

N: A parameter that denotes the size of the problem.
Master task: The Master task has the function of distributing and collecting data from the slaves as well as executing whatever serial code is necessary to solve the problem at hand. The Master is composed by:

$Seq_a(N)$: An initial sequential code probably used to generate or read some data, or to preprocess the data before the parallel algorithm starts;
$Split(N)$: The code used to divide the data to be processed among the slaves;
$Gather(N)$: The code used to collect the results from the slaves;
$Seq_b(N)$: A final sequential code, probably used to generate the final result and/or output it to its final destination.
Slave tasks: The $ST(N)$ slave tasks have the function of actually solving the problem. They are defined by:

$In(N, ST(N))$: Size of the input data for each slave;
$Out(N, ST(N))$: Size of the output produced by each slave;
$W(N, ST(N))$: The amount of work performed by the slave.
Q_i: The assigned tasks are executed by each computer c_i sequentially. And it satisfies that:

$$ST(N) = \sum_{i=1}^{P} Q_i$$

Remark 1

The items $Seq_a(N)$, $Seq_b(N)$, $Split(N)$, $Gather(N)$ and $W(N, ST(N))$ are expressed in number of operations while $In(N, ST(N))$ and $Out(N, ST(N))$ are expressed in number of data units. The following assumption appearing in Yero and Henriques (2007) are also in force in the discussion of the present paper:

H_1: $In(N, ST(N))$, $Out(N, ST(N))$ and $W(N, ST(N))$ are equal for all slave tasks, that is, the

work can be perfectly divided among the slaves.

H_2: $\lim_{N\to\infty} In(N, ST(N)) = IN < \infty$, that is, the size of the input data must not grow indefinitely with N.

H_3: $\lim_{N\to\infty} Out(N, ST(N)) = OUT < \infty$. Likewise, the size of the output data must be bounded for all values of N.

H_4: $\lim_{N\to\infty} W(N, ST(N)) = PAR < \infty$. The amount of work performed by each slave must also be bounded for all values of N.

H_5: $\lim_{N\to\infty} ST(N)\cdot W(N, ST(N)) = \infty$. The amount of work forming the parallelizable part of the problem goes to ∞ with N.

H_6: $ST(N) \geq P$, $\forall N, P$. The number of slave tasks must be greater than or equal to P for any P and any problem size, otherwise there would be unused processors.

H_7: $Seq_a(N)$, $Seq_b(N)$, $Split(N)$, $Gather(N)$, $In(N, ST(N))$, $Out(N, ST(N))$ and $W(N, ST(N))$ are continuous functions of N and $ST(N)$ for $N > 0$.

H_8: t_{op_i}, $1 \leq i \leq P$ does not depend on N, that is, an increase in problem size does not induce an increase in the time to execute an operation.

For a task distribution $Q' = \{Q_1, Q_2, ..., Q_P\}$ among processors, considering only computer c_i, we define the execution time T_i as:

$$\begin{aligned} T_i &= Seq_a(N)\cdot t_{op_1} + Split(N)\cdot t_{op_1} \\ &+ Q_i \cdot \left[In(N, ST(N))\cdot t_{c_i} + W(N, ST(N))\cdot t_{op_1} + Out(N, ST(N))\cdot t_{c_i} \right] \\ &+ Gather(N)\cdot t_{op_1} + Seq_b(N)\cdot t_{op_1} \end{aligned} \quad (1)$$

Then the time for the application to execute is given by:

$$\begin{aligned} T_{Q_i} &= \max_{1\leq i\leq P} T_i = Seq(N) + Ov(N) \\ &+ \max_{1\leq i\leq P} \left\{ Q_i \cdot \left[Par_i(N, ST(N)) + Comm_i(N, ST(N)) \right] \right\} \end{aligned} \quad (2)$$

Where

$$Seq(N) = \left[Seq_a(N) + Seq_b(N) \right] \cdot t_{op_1}$$

$$Ov(N) = \left[Split(N) + Gather(N) \right] \cdot t_{op_1}$$

$$Comm_i(N, ST(N)) = \left[In(N, ST(N)) + Out(N, ST(N)) \right] \cdot t_{c_i}$$

$$Par_i(N, ST(N)) = W(N, ST(N)) \cdot t_{op_i}$$

From Equation 2, we see that the parallel execution time $T_{Q'}$ depends on the scheduling Q' used. If Q is defined as the set of all possible scheduling decisions Q' for $ST(N)$ slave tasks considering the optimum scheduling decision, the minimum execution time $T_{\min}$ can be presented as:

$$T_{\min} = \min_{Q'\in Q}(T_{Q'}) \quad (3)$$

Consider the time T^- defined as:

$$T^- = Seq^-(N) + Ov^-(N) + \frac{ST(N)}{P}[Par^-(N, ST(N)) + Comm^-(N, ST(N))] \quad (4)$$

Where

$$Seq^-(N) = [Seq_a(N) + Seq_b(N)] \cdot t_{op}^-$$

$$Ov^-(N) = [Split(N) + Gather(N)] \cdot t_{op}^-$$

$$Comm^-(N, ST(N)) = [In(N, ST(N)) + Out(N, ST(N))] \cdot t_c^-$$

$$Par^-(N, ST(N)) = W(N, ST(N)) \cdot t_{op}^-$$

$$t_{op}^- = \min_{1\leq i\leq P}(t_{op_i}) \text{ and } t_c^- = \min_{1\leq i\leq P}(t_{c_i})$$

Furthermore, we formally define the relative speedup as:

$$S(N, P) = \frac{T_{seq}}{T_{min}} \quad (5)$$

Where $T_{\min}$ is defined by Equation 3 and

$$T_{seq} = Seq^-(N) + ST(N) \cdot Par^-(N, ST(N)) \quad (6)$$

The task allocation model

For a given problem, assume that the parallel algorithm has been already designed. The main problems are how to subdivide the task to each processor and how to carry out these tasks on the parallel platform. Obviously, different schemes will have different influence on the utilization of the computing ability to the parallel system.

We will answer the question on how to decrease the execution time. Of course, it is easy for the homogeneous clusters, but difficult for the heterogeneous cluster. So the task allocation scheme has a strong influence for the execution time. For the Master-Slave application on large heterogeneous clusters, the optimal scheduling have been considered by using the Divisible Load Theory (known as DLT) (Robertazzi, 2003) as proposed in Shylo et al. (2011), Mohamed and Al-Jaroodi (2011) and Yu and Robertazzi (2003). For a minimum time solution, all processors must stop computing at the same instant, otherwise load could be transferred from busy to idle processors. Here, optimality is defined in the context of the specific interconnection topology and scheduling policy used. In divisible load distribution theory, it is assumed that computation and communication loads can be partitioned arbitrarily among a number of processors and links, respectively. Furthermore, the theory of divisible load is fundamentally deterministic before the implementation. Firstly, the idea of the DLT is that all processors must stop computing at the same time by pre-partition. Actually, there are a large number of unpredictable factors when performing. So the theoretical optimal time may not be the optimal time in practice.

Secondly, according to the theory of DLT, the more processors lead to higher efficiency. So the DLT attempts to make every processor work at the same time. But in practice, the increase of the number of the processors may not lead to the improvement of efficiency, because assigning the task and data communication may have a great influence on the parallel computing. Sometimes, we would rather let the sub-task wait for the high-capacity processor than send it to the low-capacity processor. Thirdly, it is assumed that computation and communication load can be partitioned arbitrarily, but in practice, there are a large number of problems which cannot be partitioned arbitrarily such as the task need the pre-results. In order to solve the aforementioned problems, we like to consider a new optimal allocation task model in another point of view. We aim to let this model help us to realize the optimal execution time in practice. In the following, we construct this task allocation model to realize the real-time task distribution. The main procedures are as follows:

i) Let Master c_1 be the Master and $c_i\,(1 \le i \le P)$ be the slaves. First, the Master initializes the data and subdivides the task into $ST(N)$ sub-tasks that each sub-task is equivalent.
ii) Every slave processor receives a sub-task from the Master and executes its assigned task in the parallel way. After the first allocation action, there are $ST(N)-P$ sub-tasks left.
iii) When the slave processor finishes the sub-task, the results will be sent back to the Master processor and then the slave processor will be added into the waiting sequence to wait for the assigned task from the Master. The waiting sequence denotes as $c' = \{c'_1, c'_2, ..., c'_P\}$. If there are several Slave processors which finish the sub-tasks simultaneously, the element of waiting sequence will be more than one, that is, $P > 1$. In addition, the amount of the sub-tasks implemented by each Slave processor will be recorded on the Master processor, this is, $N_P = \{N_1, N_2, ..., N_P\}$, where N_i respects the number of the sub-tasks that have been implemented by Slave processor c_i. Thus, we can rank the waiting sequence from high to low according to the work they have implemented.
iv) The Master processor only allocates one sub-task to one Slave processor at the same time. So the first element of the waiting sequence will get the first sub-task from the Master processor. The procedure will be ended until all tasks have been implemented.

Remark 2

Now, we analyze if the aforementioned allocation algorithm can realize the minimum execution time. The idea of the aforementioned algorithm is that the best computer will perform as more sub-tasks as possible. In the macroscopical perspective, the execution time should be minimum. The model does not give the optimal allocation distribution scheduling before implementation, instead it allocates the sub-tasks to the Slave computers in the course of implementation according to the performance of each computer in practice. Furthermore, we can also use the model to solve the problem for which the loads can not be partitioned arbitrarily among a number of processors and links, respectively. The optimal allocation scheme $Q' = \{Q_1, Q_2, ..., Q_P\}$ can be got when all the tasks have been performed.

THE INFLUENCE OF THE ASSIGNED MODEL TO THE SPEEDUP

When evaluating a parallel system, we are often interested in how much performance is achieved by parallelizing a given application over a sequential implementation. Here, we will give another more rigorous upper bound of the speedup. Generally, the performance characteristics of each computer in the parallel program are transparent to the users, that is, it is impossible for us to get all the information of each computer used in the program. In other words, if the performance characteristics of each computer are known, we can have a rigorous speedup analysis based on the hardware system, but the transplant will be lost. Hence, it is much adaptive and practical to investigate the speedup bounds independent of the hardware. For the parallel system, the

minimum execution time is defined as:

$$T_{\min} = \min_{Q' \in Q} T_{Q'}$$

Where Q indicates the set of all the tasks allocation scheme. Let the optimal allocation scheme $Q' = \{Q_1, Q_2, ..., Q_P\}$ and define Q'_d as:

$$Q'_d = Q'_{\max} - Q'_{\min} \tag{7}$$

Where $Q'_{\max} = \max_{1 \le i \le P} Q'_i$ and $Q'_{\min} = \min_{1 \le i \le P} Q'_i$. Furthermore, it is the difference of the Slave with the most sub-tasks and the Slave with the least sub-tasks. Obviously, we have $Q'_{\max} = Q'_{\min}$ for the homogenous cluster system and $Q'_d \ge 0$ for the heterogeneous cluster system. Before stating the main theorem, we give the following lemma firstly.

Lemma

Let the number of Slave tasks be $ST(N)$ and the number of processors be P. If Q'_d is fixed, for a task allocation scheme Q', we have that:

$$\frac{ST(N) + Q'_d}{P} \le Q'_{\max} \le \frac{ST(N) + (P-1) \cdot Q'_d}{P} \tag{8}$$

Proof

Firstly, we give the proof of the lower bound. If Q'_d is a constant, the maximum of $ST(N)$ is $Q'_{\min} + (P-1) \cdot Q'_{\max}$. Then we have:

$$\begin{aligned} ST(N) &\le Q'_{\min} + (P-1) \cdot Q'_{\max} \\ &\le Q'_{\min} + (P-1) \cdot (Q'_{\min} + Q'_d) \\ &\le P \cdot Q'_{\min} + (P-1) \cdot Q'_d \end{aligned} \tag{9}$$

By transposition, we can get:

$$Q'_{\min} \ge \frac{ST(N) - (P-1) \cdot Q'_d}{P} \tag{10}$$

Furthermore, we obtain the lower bound of $Q'_{\max}$, that is:

$$\begin{aligned} Q'_{\max} &= Q'_{\min} + Q'_d \\ &\ge \frac{ST(N) - (P-1) \cdot Q'_d}{P} + Q'_d \\ &= \frac{ST(N) + Q'_d}{P} \end{aligned} \tag{11}$$

Similarly, the minimum of $ST(N)$ is $Q'_{\max} + (P-1) \cdot Q'_{\min}$, then we have that:

$$\begin{aligned} ST(N) &\ge (P-1) \cdot (Q'_{\max} - Q'_d) + Q'_{\max} \\ ST(N) &\ge P \cdot Q'_{\max} - Q'_d \cdot (P-1) \end{aligned} \tag{12}$$

So it follows that:

$$Q'_{\max} \le \frac{ST(N) + (P-1) \cdot Q'_d}{P} \tag{13}$$

Combining Equations 11 and 13, we get the inequality (Equation 8). Before, giving the main theorem, we add another assumption:

H9: $Ov(N) = Comm(N, ST(N)) = 0$.

Remark 3

Although, we assume $Ov(N) = Comm(N, ST(N)) = 0$, but $Seq(N)$ cannot be neglected. This is because the time of data communication between the Master and Slaves is very small compared with the whole execution time. According to the idea of the parallel algorithm in the aforementioned discussion, the parallel algorithm includes two parts: sequence computation and parallel computation. The proportions of each part are different for the different problems in practice. Thus, $Seq(N)$ should not be neglected. But for different problems, the proportions of the sequence computation part are different. And it is not easy to get the exact value of the proportion. In the proof of the following Theorem, we shall show that Lemma can help us overcome this difficulty.

Theorem

Assume H_9 holds for a fixed number of processors P and a fixed size problem N, if the number of Slave tasks is $ST(N)$ and the minimum execution time of the task allocation scheme Q' is $T_{\min}$, then the speedup of the

parallel algorithm satisfies:

$$S(N,P) \le \frac{ST(N)\cdot P}{ST(N)+Q'_d} \tag{14}$$

Proof

According to the definition of the speedup, that is $S(N,P)=\frac{T_{seq}}{T_{\min}}$, substituting the formula of T_{seq} and $T_{\min}$, that is Equations 3 and 6 into it, we arrive at:

$$S(N,P)=\frac{Seq^{-}(N)+ST(N)\cdot Par^{-}(N,ST(N))}{Seq(N)+Ov(N)+\max\limits_{1\le i\le P}\{Q_i\cdot[Par_i(N,ST(N))+Comm_i(N,ST(N))]\}} \tag{15}$$

Using the assumption H9, we have:

$$S(N,P)=\frac{Seq^{-}(N)+ST(N)\cdot Par^{-}(N,ST(N))}{Seq(N)+\max\limits_{1\le i\le P}\{Q_i\cdot Par_i(N,ST(N))\}} \tag{16}$$

Noting $Par^{-}(N,ST(N)) \le Par(N,ST(N))$, $Seq^{-}(N)\le Seq(N)$, we get:

$$\begin{aligned} S(N,P) &\le \frac{Seq^{-}(N)+ST(N)\cdot Par^{-}(N,ST(N))}{Seq^{-}(N)+\max\limits_{1\le i\le P}\{Q_i\cdot Par^{-}(N,ST(N))\}} \\ S(N,P) &\le \frac{Seq^{-}(N)+ST(N)\cdot Par^{-}(N,ST(N))}{Seq^{-}(N)+\max\limits_{1\le i\le P}\{Q_i\}\cdot Par^{-}(N,ST(N))} \end{aligned} \tag{17}$$

For $Seq^{-}(N)\ge 0$, according to the inequality $\frac{A+c}{B+c}<\frac{A}{B}$ ($A,B,c>0$), it follows that:

$$S(N,P) \le \frac{ST(N)\cdot Par^{-}(N,ST(N))}{\max\limits_{1\le i\le P}\{Q_i\}\cdot Par^{-}(N,ST(N))} \tag{18}$$

That is:

$$S(N,P) \le \frac{ST(N)}{\max\limits_{1\le i\le P}\{Q_i\}} \tag{19}$$

By the definition of $Q'_{\max}=\max\limits_{1\le i\le P}\{Q'_i\}$, it follows that:

$$S(N,P) \le \frac{ST(N)}{Q'_{\max}} \tag{20}$$

If Q'_d is known, according to Lemma, we have $Q'_{\max}\ge\frac{ST(N)+Q'_d}{P}$. Substituting this into Equation 20, we get:

$$S(N,P) \le \frac{ST(N)}{\frac{ST(N)+Q'_d}{P}} \tag{21}$$

That is $S(N,P)\le\frac{ST(N)\cdot P}{ST(N)+Q'_d}$

Which is what we intend to prove.

Obviously, the Theorem implies $ST(N)\le P$ if and only if $Q'_d=0$, which says this heterogeneous cluster system transfers to a homogenous cluster system if there is no difference between $Q'_{\max}$ and $Q'_{\min}$. Here, we rewrite Equation 14 as:

$$S(N,P) \le \delta\cdot P \tag{22}$$

Where $\delta=\frac{ST(N)}{ST(N)+Q'_d}$. We can find that, the homogenous cluster system leads to $Q'_d>0$ which makes $\delta<1$. So the homogenous cluster system decreases the lower bound of the speedup of the system. And this can be measured by the difference between the "best" processor and the "worst" processor, namely Q'_d.

EXAMPLE OF PARALLEL COMPUTATION

There are rich literatures which present the naive parallel matrix multiplication algorithm (Beaumont et al., 2001; Li, 2001, 2010). But in this section, we will present a naive parallel implementation of the classical problem; this is the prime number searching. The method of sifting the prime numbers is popular among the methods to solve this problem. The basic idea is as follows:

1) Generate a natural number sequence $A_n=\{2,3,\ \ldots,n\}$;

2) Delete all multiples of 2, 3, 4, …, $\left[\frac{n}{2}\right]$ but the number

Table 1. Information of the hardware.

Computer	CPU	EMS memory (M)
Master	Celeron (R), 2.80 GHz	1024M
S1	AMD Athlon (tm); dual core processor 4000+, 1.8 GHz	512 M
S2	Intel (R) T2080 1.73 GHz	1024 M
S3	AMD Athlon (tm); dual core processor 4600+, 2.4 GHz	1.5 GM

itself, where $\left[\frac{n}{2}\right]$ is the floor of $\frac{n}{2}$.

Generalizing the idea, we can search the prime number from m to n, where $m<n$. The idea is as follows:

1) Generate a natural number sequence $A_n=\{m, m+1, \ldots, n\}$;

2) Delete the multiples of 2, 3, 4, …, $\left[\frac{n}{2}\right]$, this is, generally, generate a sequence:

$$B_i=\left\{i\times t \mid i\times t\in A_n, \forall t\in N^+, \forall i\in\left\{2,3,\ldots,\left[\frac{n}{2}\right]\right\}\right\}$$

and obtain its complement $A_n-\bigcup_i B_i$.

To set up the heterogeneous cluster system, we consider a system which is composed of one Master processor and three Slave processors in the experiment. The information of each computer is given in Table 1.

Sifting prime number with Matlab

According to the algorithm described earlier, the codes for sifting prime numbers are as follows:

```
1) Function (R, time) = Prime (m, n);
2) syms time;
3) Time = cputime;
4) A = (m: n);
5) For I = 2:floor (n/2);
6) mint = max(ceil(m/i),2);
7) maxt = floor(n/i);
8) Clear B;
9) B = i*(mint:maxt);
10) For j = 1: size (B, 2);
11) A (find (A==B(j))) = [ ];
12) End;
13) End;
14) A (find (A==1)) = [ ];
15) R = A;
16) Time = cputime – time.
```

The next work is to design the main function whose jobs are to manage the system, initialize the data, allocate the task, communicate the information, etc. Subdivide the problem n into $ST(N)$ sub-tasks as:

$\text{Prime}(1, \frac{n}{ST(N)})$, $\text{Prime}(\frac{n}{ST(N)}+1, \frac{2n}{ST(N)})$, …, $\text{Prime}(\frac{n(ST(N)-1)}{ST(N)}+1, n)$.

The codes of the main function are as follows:

```
1) Function ParallelPrime.
2) tic;
3) jm = find RESOURCE ('scheduler', 'type' ,'jobmanager', 'name', 'master computer', 'LookupURL', '192.168.0.101');
4) Job = createJob (jm);
5) Set (job, 'File Dependencies', {'Prime.m'}),
6) MAXNUM = 50000; m (1: 10) = [0]; n (1:10) = [0];
7) For I = 1:10,
8) m (i) = 1 + (I - 1) * (MAXNUM./10);
9) n (i) = i* (MAXNUM./10);
10) End;
11) Create Task (job, @Prime, 1, {m (1), n (1)});
12) Create Task (job, @Prime, 1, {m (2), n (2)});
13) Create Task (job, @Prime, 1, {m (3), n (3)});
14) Create Task (job, @Prime, 1, {m (4), n (4)});
15) Create Task (job, @Prime, 1, {m (5), n (5)});
16) Create Task (job, @Prime, 1, {m (6), n (6)});
17) Create Task (job, @Prime, 1, {m (7), n (7)});
18) Create Task (job, @Prime, 1, {m (8), n (8)});
19) Create Task (job, @Prime, 1, {m (9), n (9)});
20) Create Task (job, @Prime, 1, {m (10), n (10)});
21) submit (job);
22) Wait For State (job, 'finished');
23) Sub results = get All Output Arguments (job);
24) Results = [ ];
25 For i = 1: size (subresults, 1);
26) Results = (results, sub results {i});
27) End;
28) Toc;
```

These are the codes of the parallel algorithm. By definition

Table 2. Record of the parallel computation.

Size (N)	Time (s)	Size (N)	Time (s)
50000	8.39562867	130000	31.0759633
60000	9.97009067	140000	34.4556117
70000	12.1911177	150000	38.6891003
80000	14.7904617	160000	43.33529
90000	17.455761	170000	47.7182023
100000	20.016397	180000	53.7299207
110000	23.4023313	190000	57.8931747
120000	26.8131993	200000	63.5699997

of speedup, the experiment needs to be designed with the corresponding sequential algorithm. But there are two forms about the sequential algorithm, one is to run the function $Prime(1,MAXIMUM)$ on one computer directly, the other is to subdivide the task into $ST(N)$ sub-tasks like the parallel computation and run them one by one on one computer. So in this experiment we design two sequential algorithms to analyze the speedup. For convenience, we call the first sequential algorithm the 'series algorithm' and call the second sequential algorithm the 'dispersal algorithm'. Using the function $Prime(m,n)$, the codes of sequential algorithm are as follows:

The codes of the 'series algorithm':

```
1) Function SeqPrime;
2) clc;
3) tic;
5) MAXNUM = 50000;
6) Prime (1, MAXNUM);
7) toc;
```

The codes of the 'dispersal algorithm':

```
1) Function SeqPrime_2;
2) clc;
3) syms Seqtime;
4) Seqtime = 0;
5) Results = [ ];
6) MAXNUM = 20000;
7) for I = 0:9;
8) m = 1 + i* (MAXNUM./10);
9) n = (I + 1)* (MAXNUM./10);
10) (R, T) = Prime (m, n);
11) Results = (Results, R);
12) Seqtime = Seqtime+T;
13) End.
```

We list the experiment records next; then, we analyze and discuss the obtained experiment results.

EXPERIMENT RESULTS

For the comprehensive analysis of the problem, we design the following three experiments: parallel computation, series computation and dispersal computation.

Computation of parallel algorithm

This experiment aims to analyze the relation between the size of the problem and the execution time (Table 2). If we fit the aforementioned data with polynomial, the polynomial is $y = 0.1282x^2 + 0.4958x + 2.5431$, $R^2 = 0.9998$ which indicates the goodness of fit. The results shown in Figure 1 demonstrate that if the size of the problem increases, then the execution time increases nonlinearly.

Computation of dispersal algorithm

In order to compare with the parallel system, the size of the problem is the same as the parallel research; so the data of the experiment given in Table 3. Similarly, we can find a function to fit the data $y = 0.2366x^2 + 3.6926x + 9.8107$, $R^2 = 1$ which indicates the goodness of fit. Furthermore, we also observe that the increasing of the execution time is nonlinear with the problem scale increasing (Figure 2).

Computation of series algorithm

The idea of the series algorithm is to search the prime from 0 to N directly. From our experience, the EMS memory of the computer has an influence on the execution time as the size of the problem increases. The experiment data are given in Table 4. To fit the data, we find the function $y = 0.2145x^3 - 1.5994x^2 + 3.6621x + 2.7912$,

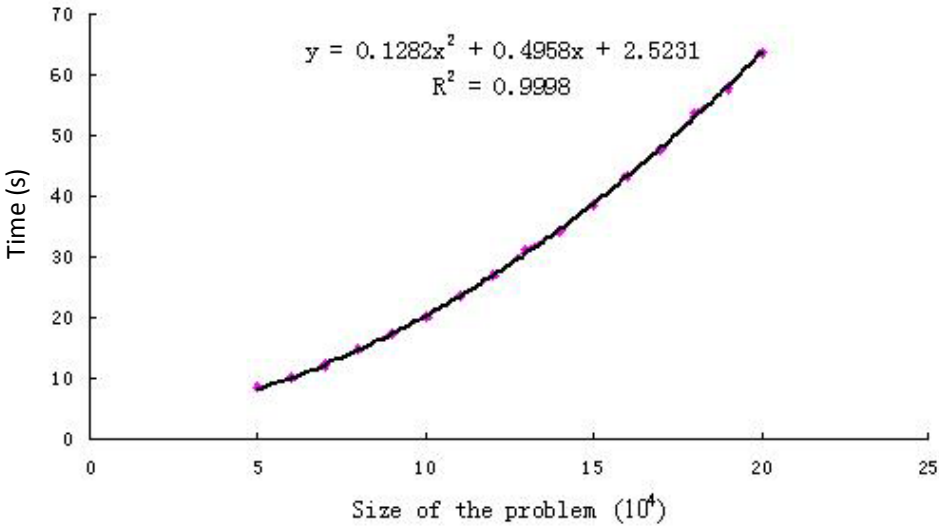

Figure 1. The curve of the parallel algorithm.

Table 3. Record of the dispersal computation.

Size (N)	Time (s)	Size (N)	Time (s)
50000	14.0781	130000	62.5
60000	18.1719	140000	70.3438
70000	23.0156	150000	78.8594
80000	28.1719	160000	88.5469
90000	33.9064	170000	98.2656
100000	40.1094	180000	107.8438
110000	47.125	190000	118.3906
120000	54.6563	200000	129.0938

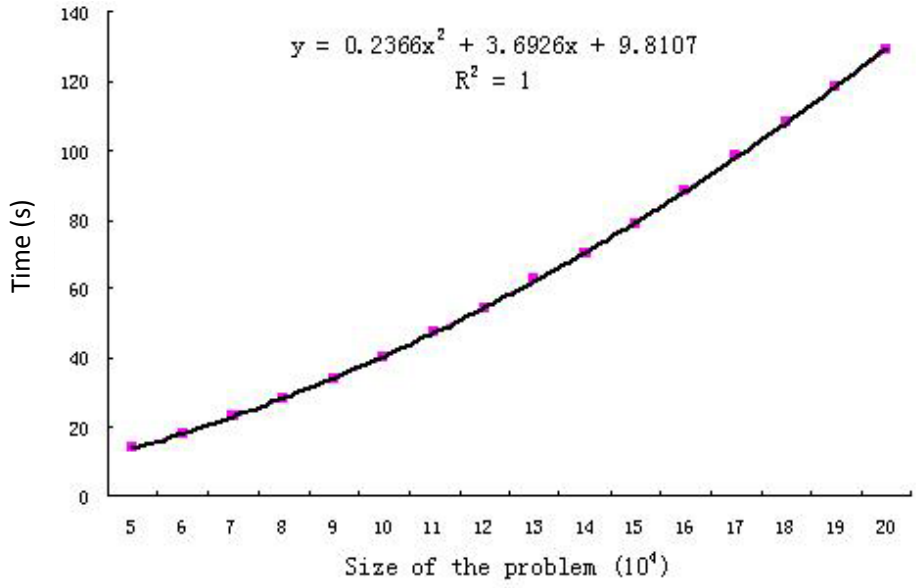

Figure 2. The curve of the dispersal algorithm.

Table 4. Record of the series computation.

Size (N)	Time (s)	Size (N)	Time (s)
10000	3.0156	40000	38.9844
15000	6.2813	50000	62.1563
16000	7.0313	60000	94.2813
17000	7.875	70000	134.7031
18000	8.6094	80000	190.0781
20000	10.35945	90000	252.875
30000	22.375	10000	326.9531

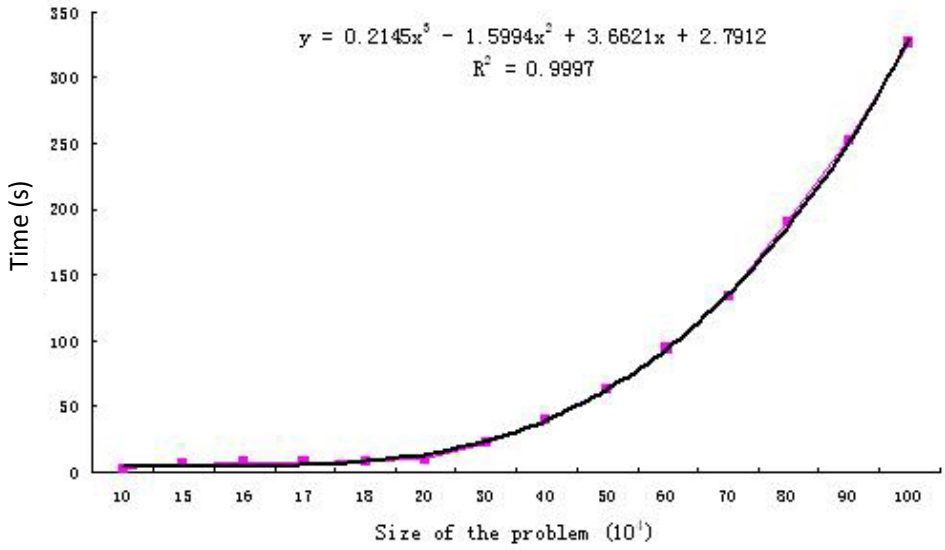

Figure 3. The curve of the series algorithm.

$R^2 = 0.9997$ which indicates the goodness of fit. Figure 3 illustrates that this algorithm has a strong nonlinear increasing trend which means the algorithm is not very stable. That means if the size of the problem is large enough, the system may not be able to finish the task in finite time.

The results

Figure 4 demonstrates that the nonlinear characteristic of the series is stronger than the dispersal algorithm. The main reasons are that the EMS memory plays an important role in the computation. For a large size problem, a large proportion of the EMS memory will be occupied for data stored and processed. Just very limited EMS memory is left for running the algorithm. Thus, the execution time will increase rapidly even unpredictably. The dispersal algorithm can perform well because the EMS memory is enough for every computation. This is also the reason; for that its nonlinear characteristic is weaker:

1) The execution time of the series algorithm is unpredictable as the size of the problem increases, thus, the dispersal algorithm is relatively stable.
2) The efficiency of solving the problem in unite time will decrease as the size of the problem increases.
3) As shown in Figure 4, the resource of the computer will have a limit to process the series algorithm, that is, the series algorithm is not suitable for the large-scale problems.
4) For a large problem, partitioning the problem is a helpful way to optimize the algorithm and will increase the efficiency of the sequential algorithm.

DISCUSSION

Analysis of the results

From Figure 5a, comparing the nonlinear characteristic of

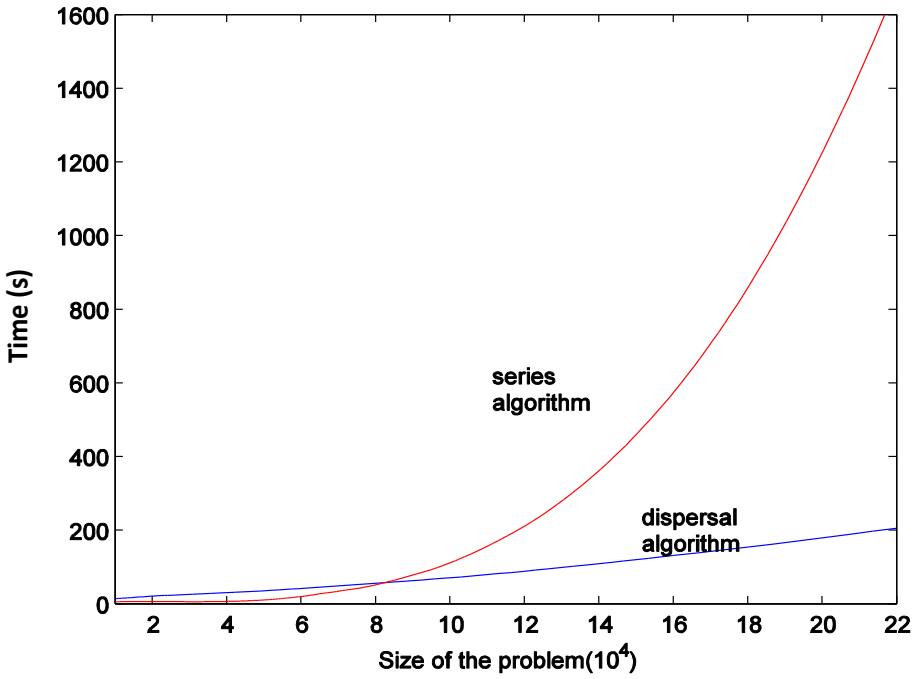

Figure 4. Comparison of the series and dispersal algorithm.

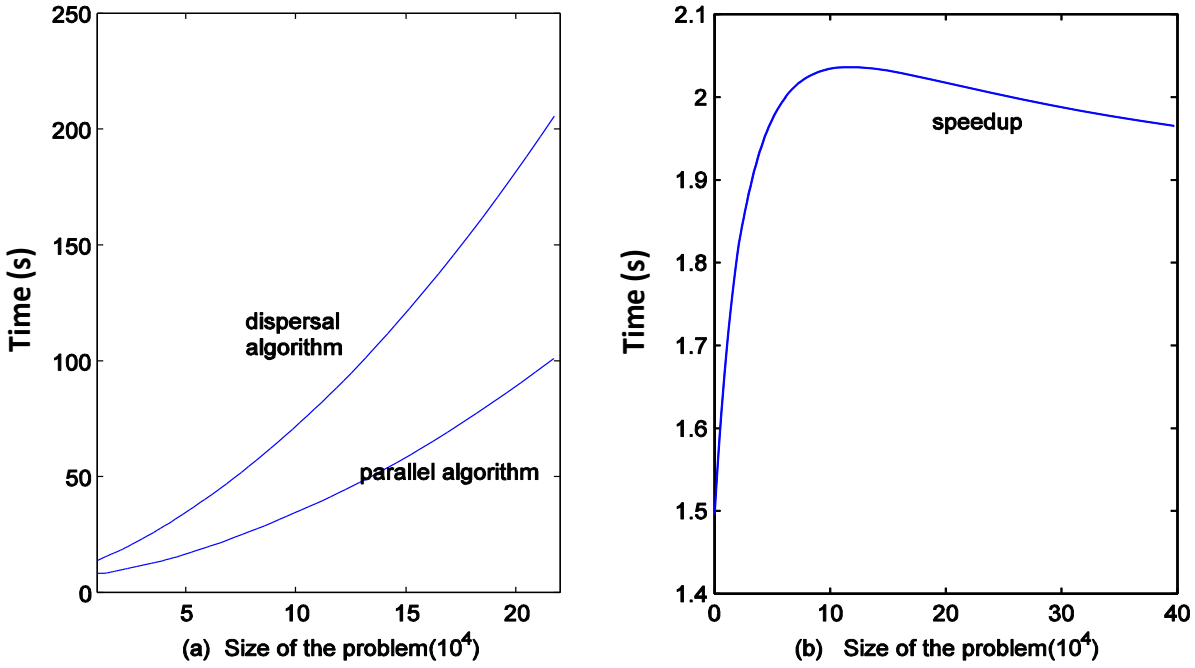

Figure 5. Speedup and the comparison of the dispersal and parallel algorithm.

the parallel with the sequential algorithm, we conclude that the nonlinear characteristic of the parallel algorithm is weaker than the sequential algorithm, that is, the parallel algorithm is more stable. The reason is that the influence of all the negative factors will be assigned to each computer in the parallel frame. Figure 5b shows the speedup which is the relative speedup defined earlier. We see an interesting phenomenon that the speedup

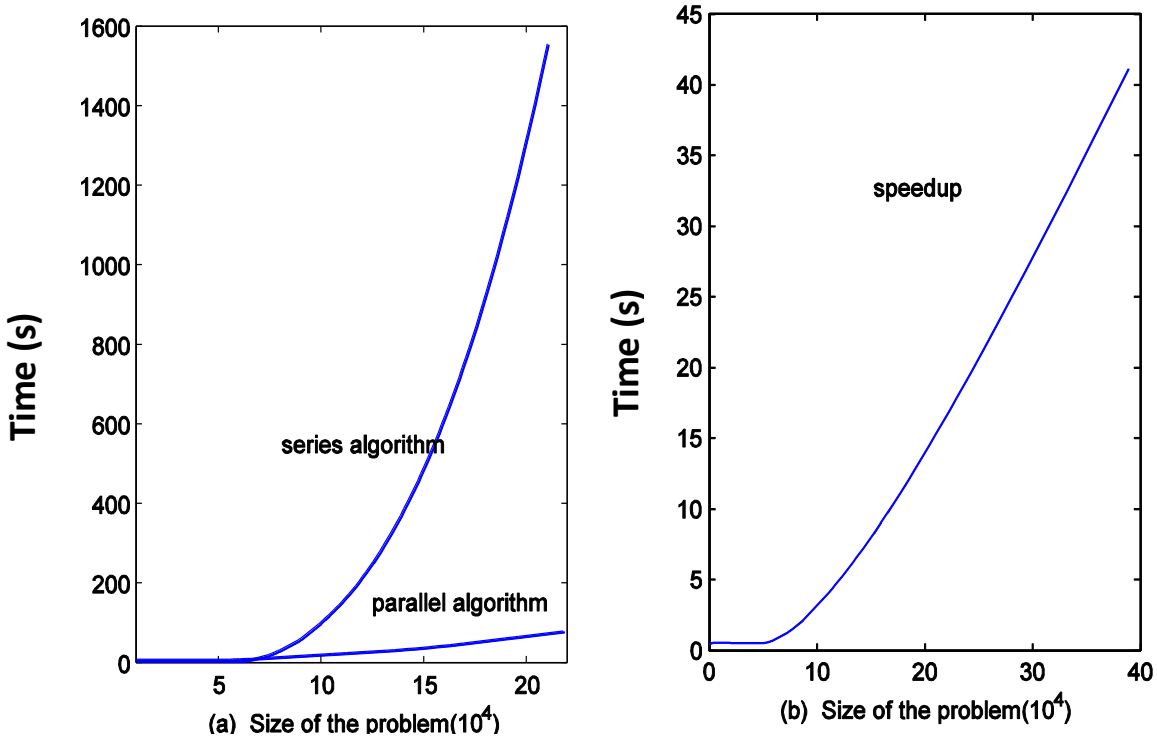

Figure 6. Speedup and the comparison of the series and parallel algorithm.

approaches its maximum rapidly and then decrease to a constant slowly as the problem size increases:

1) The proportion of the time for the data communication and task partition is relatively large for a small enough problem.
2) The maximum value indicates that at this special problem size level, the efficiency of the performance is the highest for the parallel system. By simple computation, we get the maximum value 2.0381 at $N = 119963$. This implies that however large the problem is, the size of the Slave task should be arranged to be 119963 in order to get the best efficiency.
3) The speedup has a limit when the size of the problem $N \to \infty$. This is to say the parallel system is stable as the size increases.
4) It verifies that the upper of the speedup is smaller than the number of the computers of the cluster, that is, Speedup $< P = 3$. This also verifies the conclusion of the Theorem.

DISCUSSION AND CONCLUSION

Under the classical definition of the speedup, that is the ratio of the execution time of parallel algorithm to series algorithm, as shown in Figure 6b, we obverse that the speedup will increase nonlinearly and rapidly, furthermore, it is more than the number of the processors of the cluster. We call this phenomenon the super-speedup which is different from the existing works in Yero and Henriques (2007). Compared with the results in Yero and Henriques (2007), the reason of this phenomenon is that the necessary partition of the task optimizes the processing and in some sense the partition have the signification of 'speedup'. In practice, we optimize the processing twice including partition of the task and the parallel computing. Thus, the speedup is the superposition of these two phenomena. Of course, it will exceed the speedup caused by single factor which is the parallel computing. This is the reason that the speedup is larger than P. But there is no contradiction with the main conclusions aforementioned. On the contrary, it demonstrates the phenomenon of the super-speedup and it is caused by the experiment designs.

ACKNOWLEDGEMENTS

This work was supported by the National Natural Science Foundation of China (11101102), Ph.D. Programs Foundation of Ministry of Education of China (20102304120022), the Natural Science Foundation of Heilongjiang Province (A201014), Foundational Science Foundation of Harbin Engineering University and Fundamental Research Funds for the Central Universities (HEUCF20111101).

REFERENCES

Almeida F, González D, Moreno LM (2006). The master-slave paradigm on heterogeneous systems: a dynamic programming approach for the optimal mapping. J. Syst. Architecture, 52(2): 105-116.

Amdahl G (1967). Validity of the single-processor approach to achieving large scale computing capabilities. Proc. AFIPS, pp. 483-485.

Bastian P, Blatt M, Dedner A, Engwer C, Klöfkorn R, Kornhuber R, Ohlberger M, Sander O (2008). A generic grid interface for parallel and adaptive scientific computing. Part II: implementation and tests in DUNE. Comput., 82(2-3): 121-138.

Beaumont O, Boudet V, Rastello F, Robert Y (2001). Maxtri multiplications on heterogeneous platforms. IEEE Trans. Parallel Distrib. Syst., 12(10): 1033-1051.

Dutta H, Fiorletta H, Pooleery M, Diab H, German S, Waltz D, Schevon CA (2011). A case-study on learning from large-scale intracranial EEG data using multi-core machines and clusters. Proc. LDMTA, pp. 1-8.

Gonzalez-Velez H, Cole M (2008). An adaptive parallel pipeline pattern for grids. Proc. IPDPS, pp. 1-11.

Goswami P, Makhinya M, Bösch J, Pajarola R (2010). Scalable parallel out-of-core terrain rendering. Proc. of ESPGV, pp. 63-71.

Gustafson J (1988). Reevaluating Amdahl's law. Comm. ACM, 31: 532-533.

Head MR, Govindaraju M (2007). Approaching a parallelized XML parser optimized for multi-core processors. Proc. SOCP: 17-22.

Head MR, Govindaraju M (2009). Performance enhancement with speculative execution based parallelism for processing large-scale xml-based application data. Proc. ACM HPDC: 21-30.

Li K (2001). Scalable parallel matrix multiplication on distributed memory parallel computers. J. Parallel Distrib. Comput., 61: 1709-1731.

Li K, Pan Y, Shen H, Zheng SQ (1999). A study of average-Case speedup and scalability of parallel computations on static networks. Math. Comput. Model, 29: 83-94.

Mohamed N, Al-Jaroodi J (2011). Delay-tolerant dynamic load balancing. Proc. HPCC: pp. 237-245.

Robertazzi T (2003). Ten reasons to use divisible load theory. Comput., 36(5): 63-72.

Sanjay HA, Vadhiyar S (2008). Performance modeling of parallel applications for grid scheduling. J. of Parallel and Distrib. Comput., 68(8): 1135-1145.

Shylo OV, Middelkoop T, Pardalos PM (2011). Restart strategies in optimization: parallel and serial cases. Parallel Comput., 37(1): 60-68.

Silva da FAB, Senger H (2011). Scalability limits of Bag-of-Tasks applications running on hierarchical platforms. J. Parallel Distrib. Comput., 71(6): 788-801.

Snyder L, Sterling T (2000). Panel: "What are the top ten most influential parallel and distributed processing concepts of the last millennium?". J. Parallel. Distrib. Comput., 61(12): 1827-1841.

Williams R (2011). Parallelizing time with polynomial circuits theory. Comput. Syst., 48: 150-169.

Yan Y, Zhang X, Song Y (1996). An effective and practical performance prediction model for parallel computing on nondedicated heterogeous now. J. Parallel Distrib. Comput., 38(1): 63-80.

Yang XJ, Du J, Wang ZY (2011). An effective speedup metric for measuring productivity in large-scale parallel computer systems. J. Supercomput.., 56(2): 164-181.

Yero EJH, Henriques MAA (2007). Speedup and scalability analysis of Master-Slave appllications on large heterogeneous clusters, J. Parallel Distrib. Comput., 67: 1155-1167.

Yu D, Robertazzi T (2003). Divisible load scheduling for grid computing. Proc. PDCS, pp. 1-6.