An Application-Level Scheduling with Task Bundling Approach for Many-Task Computing in Heterogeneous Environments

Jian Xiao, Yu Zhang, Shuwei Chen, Huashan Yu

An Application-level Scheduling with Task Bundling Approach for Many-Task Computing in Heterogeneous Environments

Jian Xiao, Yu Zhang, Shuwei Chen, Huashan Yu*

School of Electronic Engineering and Computer Science, Peking University
Beijing 100871, P.R. China
{xiaojian, zhangyu, csw}@net.pku.edu.cn,yuhs@pku.edu.cn

Abstract. Many-Task Computing (MTC) is a widely used computing paradigm for large-scale task-parallel processing. One of the key issues in MTC is to schedule a large number of independent tasks onto heterogeneous resources. Traditional task-level scheduling heuristics, like Min-Min, Sufferage and MaxStd, cannot readily be applied in this scenario. As most of MTC tasks are usually fine-grained, the resource management overhead would be prominent and the multi-core nodes might become hard to be fully utilized. In this paper we propose an application-level scheduling with task bundling approach that utilizes the knowledge of both applications and tasks to overcome these difficulties. Furthermore we adapt the traditional task-level heuristics to our model for MTC scheduling. Experimental results show that these application-level scheduling approaches, when equipped with task bundling, can deliver good performance for Many-Task Computing in terms of both Makespan and Flowtime.

Keywords: Keyword. application-level scheduling, many task computing, task bundling, traditional scheduling heuristics

1 Introduction

Many-task Computing (MTC) [1] [2] is a loosely coupled computing paradigm that is widely used for scientific applications such as parameter sweep, Monte Carlo simulations, data parallelism, bioinformatics (like BLAST) and image manipulation [2], and the major goal of which is to complete a large number of independently-schedulable tasks within a short period of time. The application in this paradigm is usually developed upon abundant legacy executables accumulated in scientific domain. A scientific computing problem is represented as a MTC job, which could contain as many as thousands to millions fine-grained tasks, and thus the computation complexity could be extraordinarily demanding. The developments of large-scale processing techniques such as Supercomputer, Grid and Cloud in recent years have made it possible for MTC jobs to get results in a reasonable timeframe [1].

To efficiently execute MTC tasks in these large-scale heterogeneous environments, task scheduling is a key issue. However, scheduling MTC tasks is not a trivial

problem. The difficulty of MTC scheduling comes from not only the heterogeneity and dynamicity of resources, but also the large number of relatively-small tasks, which the latter might result in huge scheduling cost and starving a lot of computing nodes, and result in severe resource waste for multi-processor computers.

In this paper, we propose an application-level scheduling with task bundling solution to this problem. The proposed approach divides the scheduling process into two stages. In the first stage, the algorithm works at application and job level, matching job and resource by application-resource pairing and job selection, so as to improve the job's computing performance and resource's utilization simultaneously; then in the second stage, the algorithm works at task level to select a package of tasks (task selection) from the previously selected job and then allocates the package to the selected resource in one dispatch, so as to amortize the resource management overhead among tasks in same package and meanwhile exploit the intra-node capacity.

The remaining of the paper is organized as follows. Section 2 describes the problem. Section 3 reviews related works. In Section 4 we present our scheduling approach. Section 5 presents the experimental results. Finally Section 6 concludes the paper.

2 Problem Statement

Terminologies in this paper are used as in [3]. An application refers to a type of jobs, while a job is an instance of some application. Instances/jobs of the same application run the same program and use the same data resources, but read different inputs. And a job is a collection of independent tasks. We will use the terms "application" and "job type", "application instance" and "job" interchangeably.

A typical example of MTC application is Genome Alternative Splicing [4] in bioinformatics that predicts a genome's all possible transcriptomes. An application instance/job in this context refers to an actual genome submitted by some user intending to find the genome's transcriptomes, and a task is a run of the program to search all possible transcriptomes of a single gene. Generally a genome consists of tens of thousands of genes, which means such a job may contain tens of thousands of independent tasks. In our practice, 70% or more of tasks in a typical job was completed within 1 second [5].

We assume multiple applications $A_1, \dots, A_p$, each of which may contain several jobs. Applications are heterogeneous, which means every application has its own pattern of computing-resource requirement, preferring such as I/O or CPU speed, and this constitutes the application-resource knowledge to utilize.

In our work we consider a large-scale heterogeneous system $\mathfrak{R} = \{R_1, \dots, R_m\}$, consisting of network-connected single- or multi-processor computing nodes, for these applications processing. Processors of each computing node in $\mathfrak{R}$ are homogeneous, providing equal processing speed for the jobs of the same type, whereas processors from different nodes are heterogeneous.

2.1 Job model and task model

A job is a large collection of independent and relatively-small tasks sharing the same computing-resource requirement. The k-th instance of application A_i is denoted as $J_{i,k} = \{\tau_{i,k}^{(h)} : h = 1, \ldots, n_{i,k}\}$, k=1,2,.., N_i, where N_i is the job number of application A_i, $n_{i,k}$ is the task number of job $J_{i,k}$ and $\tau_{i,k}^{(h)}$ is a task contained in this job. Conventionally, the estimated execution time of each task $\tau_{i,k}^{(h)}$ is supposed to be available in advance.

2.2 Scheduling many tasks as a challenge

To schedule MTC tasks in a large-scale heterogeneous system would be a real challenge, as two traditionally negligible factors become prominent now:

- **Resource management overhead**. For a computing node to execute task, it should be allocated and configured first, and be released after the computation. The time cost of this resource management would be significant, compared with a single MTC task's computing cost itself.
- **Intra-node resource utilization.** MTC tasks are often the small-scale parallel or even sequential programs which require only few resources to complete, so a single MTC task is unable to fully exploit the power of the today's multi-core computing node, and thus cause intra-node resource waste.

2.3 Objectives

This work aims to make a trade-off between Makespan and Flowtime, so as to optimize these two competing objectives [6] simultaneously.

We denote interval $[s_{i,j}^{(k)}, e_{i,j}^{(k)}]$ as the k-th period when job i occupied node j, from time $s_{i,j}^{(k)}$ to $e_{i,j}^{(k)}$. This means that node j is processing the tasks from job i during this period, and there may be several non-intersect time periods for this job on the same node. The Makespan and Flowtime of job i are given by

$$makespan_i = \max_j \{\max_k \{e_{i,j}^{(k)}\}\} \quad (1)$$

$$flowtime_i = \sum_j \sum_k cn_j \times (e_{i,j}^{(k)} - s_{i,j}^{(k)}) \quad (2)$$

where cn_j is the core number of R_j.

Based on these, three metrics are derived for performance comparisons:

- Overall Makespan: the maximum of all $makespan_i$, used for measuring overall computing performance.
- Overall Flowtime: the sum of all $flowtime_i$, used for measuring the CPU cost to achieve the overall computing performance.

- Average Job Makespan: the average of all $makespan_i$, used for measuring the QoS of the scheduling.

3 Related Work

As a well-abstracted model, optimal scheduling of tasks for multiple processors is a NP-complete problem [7], and thus many heuristics and meta-heuristics are proposed in literatures. Heuristics are directly designed for tasks scheduling, such as MinMin [8], Sufferage [9] and MaxStd [10], while meta-heuristics are combinatorial optimization techniques used indirectly for task scheduling, and the representative meta-heuristics include Genetic Algorithm (GA) [8], Simulated Annealing (SA) [8], Particle Swarm Optimization (PSO) [11] and Chemical Reaction Optimization (CRO) [6]. These algorithms work at task level, and require the prediction of each task's execution time on each machine, forming an ETC (Expected Time to Compute) matrix [8].Though both heuristics and meta-heuristics are classic solutions to traditional HTC (High Throughput Computing) [1], they are not suitable for MTC. The drawback is, when scheduling hundreds of thousand or even more tasks to large-scale resources, the overhead would be overwhelming. What's more, as these algorithms neglect the cost for resource management, the task-level scheduling for MTC may cause a great waste in repeatedly creating and releasing runtime environment.

Several research works consider multiple applications scheduling in context of Bag-of-Tasks (BoT) [12] or cloud computing [3], but mainly focus either on ETC based task-level scheduling or homogeneous cluster environment. And these works pay litter attention to the applications' differences in resource requirements, which is an important knowledge for scheduling optimization.

Few researches exist on task bundling strategy that dispatches a package of tasks rather than a single task to the resource at every scheduling event. In [13], fixed number of tasks bundling policy is used, but it is too simple for practical use. An improved strategy proposed in [14] bundles the tasks according to the size of input file and tries to balance the task packages in terms of total input file size in the package.

Some recent works concern not only the performance but also the resource cost for this performance. In [6] and [11], meta-heuristics CRO and PSO are adopted to minimize Makespan and Flowtime simultaneously. These works verified the value of meta-heuristics for multi-objective optimization. However, the applicable conditions of these meta-heuristics are confined to small number of large tasks and negligible runtime environment preparation time (compared to task granularity).

4 Scheduling Algorithm

Our scheduling algorithm is performed in a centralized manner. The computing node sends a task request to the scheduler whenever it has no task to make a local allocation; the scheduler processes the request by assigning one or more tasks to the computing node.

Figure 1 shows the algorithm's framework. Firstly, a job-resource pair is determined, indicating the job which the computing node is allocated to; then a subset of tasks from the selected job is allocated to the computing node in one dispatch. The algorithm exploits the following knowledge about applications and tasks.

Scheduling framework
1($J_{i,k}$, R_j) ←Job-resource matching
2 Task-package B←Task-bundling ($J_{i,k}$, R_j)
3 Dispatch task package B to R_j

Fig. 1. Scheduling framework

• **Application's parallel degree, $D_{i,j}$**. Application-resource knowledge, reflecting the match between application's intrinsic nature in I/O or CPU preference and the resource's system architecture. Due to the I/O bottleneck, when the processor number of a node exceeds some limit, giving more processors will not help to improve parallel speed. We denote this limit as $D_{i,j}$, the degree of parallelism of application A_i on node R_j. $D_{i,j}$ is an integer between 1 and the processor number cn_j of R_j, represents the *logical processor* number that R_j offers to A_i and can be obtained by historical execution log.

• **Application's processing rate, $r_{i,j}$**. The other kind of application-resource knowledge. We denote $r_{i,j}$ as the processing rate of each R_j's logical processor for application A_i. This concept comes from the fact that, for any two computing nodes and different tasks from the same application, the ratio of each task's execution times on the two nodes will be quite stable and close to each other. To get the $r_{i,j}$ matrix, we can choose a *reference node*, say R_1, and let $r_{i,1}=1$; for $j \neq 1$, the $r_{i,j}$ is the ratio of Benchmark task's execution time on node R_1 and R_j.

• **Task's estimated execution time**. This is the typical task-resource knowledge, usually denoted as the ETC matrix in literatures. We define the *task's estimated complexity* as the estimated execution time on the reference node. So given the task's estimated complexity, the task's estimated execution time on any other node can be derived by using the application's processing rate matrix. We use task's estimated complexity to measure the task's granularity and workload.

4.1 Job-resource matching

The job-resource matching stage will select a job and a node from the candidates, and allocate the chosen node to the job. We consider both the requesting node's (system utilization) and the application's (computing performance) perspectives, so as to optimize the Makespan and Flowtime simultaneously.

1) Application's perspective

Given the requesting node R_j, all candidate applications are competing for this resource. We use the Node Importance (NI) to quantify how important R_j is for application A_i to achieve high computing performance, which is determined by

$$NI_{i,j} = \sum_{k=1}^{m} \omega_{i,k} \times \frac{r_{i,j} - r_{i,k}}{\overline{r_{i,.}}} \tag{3}$$

where $\overline{r_{i,.}}$ is mean value of the i-th row of processing rate matrix and $\omega_{i,k}$ is the weight that satisfies $\omega_{i,1} + \cdots + \omega_{i,m} = 1$ for any i. In this definition, $r_{i,j} - r_{i,k}$ is similar to the suffer value in Sufferage heuristic in traditional HTC, and it measures the importance gain in allocating A_i to R_j when comparing the processing rate offered by R_j with that offered by R_k, and $\frac{r_{i,j} - r_{i,k}}{\overline{r_{i,.}}}$ is the normalization of this value. If $r_{i,j} > r_{i,k}$, it is more beneficial to get R_j rather than R_k for application A_i, so the importance gain in this comparison is positive; otherwise the gain will be negative. And thus the NI value of R_j for A_i shall be the sum of importance gains in all those possible comparisons. When sum these values up, we follow the intuition of traditional HTC heuristic Sufferage, which gives larger weight to the sufferage incurred from the failure to choose better option(actually it is a 0-1 weight assignment, 1 for the difference between the best option and the second best one, and 0 for others), but adopt a smoothed weight assignment policy:

$$\omega_{i,k} = r_{i,k} / \sum_{l=1}^{m} r_{i,l} \tag{4}$$

For a requesting node R_j, each application A_i will measure it by NI value; If $NI_{i,j} > NI_{k,j}$, R_j is more important to A_i than A_k, and we should allocate A_i to R_j.

2) Requesting node's perspective

Similar to the above case, but the importance now is measured in terms of resource utilization (described by the parallel degree). Specially, when an application is going to choose a computing resource, we use Application Importance (AI) to quantify how important is A_i for R_j to achieve high resource utilization, which is given by

$$AI_{i,j} = \sum_{k=1}^{p} \lambda_{k,j} \times \frac{D_{i,j} - D_{k,j}}{\overline{D}_{.,j}} \tag{5}$$

$$\lambda_{k,j} = D_{k,j} / \sum_{l=1}^{p} D_{l,j} \tag{6}$$

where $\overline{D_{.,j}}$ is the mean value of the j-th column of parallel degree matrix, and $\lambda_{k,j}$ is the weight similar to $\omega_{i,k}$. When $AI_{i,j}$ is larger than $AI_{i,k}$ for two competing resources R_j and R_k, A_i is more important to R_j, so we should allocate R_j to A_i.

3) The global importance value

After determining the AI and NI, a global importance (GI) value is calculated for every possible pair A_i and R_j, so as to support the application-resource matching. This value is given by

$$GI_{i,j} = \mu_{i,j} NI_{i,j} + (1 - \mu_{i,j}) AI_{i,j} \tag{7}$$

where $0 \le \mu_{ij} \le 1$, reflecting the trade-off between the optimization of computing performance (Makespan) and utilization (Flowtime). We provide each GI_{ij} a private weight to accommodate more flexibility. To this end, we measure for application A_i that how spread out the processing rates it achieves on different resources is, and for node R_j that how spread out the utilization it achieves in processing different applications is. So the weight value is given by

$$\lambda_{i,j} = r\,\mathrm{cv}_i / (r\,\mathrm{cv}_i + c\,\mathrm{cv}_j) \tag{8}$$

where rcv_i and ccv_j are the Coefficient-of-Variance (CV) values of the i-th row of processing rate matrix $r_{i,j}$ and j-th column parallel degree matrix $D_{i,j}$ respectively. This weight assignment policy ensures that more opportunities are given to the side whose achievable processing rates or utilization ratios are more dispersed, and it is reasonable to prioritize those decisions.

Job-resource matching

1For every candidate application-resource pair (A_i, R_j)
2 Compute global importance value $GI_{i,j}$;
3 (A_i^*, R_j^*)←argmax{$GI_{i,j}$}; //the pair whose GI is maximal
4 J_{pre}← previously processed job of R_j^*
5 If ($J_{pre} \neq \emptyset$)
6 return pair (J_{pre}, R_j^*);
7 Else
8 $J_{i,k}^*$← job with least total complexity over all jobs of A_i^*
9 return pair ($J_{i,k}^*$, R_j^*);

Fig. 2. Job-resource Matching Algorithm

Given an application-resource pair, the rest of the job-resource matching work is just to select an instance of the targeted application (job selection). Figure 2 shows the whole job-resource matching algorithm. The time complexity is $O(mp(m+p)+N_i)$, but in practice we can pre-compute the GI values to reduce the complexity to just $O(N_i + mp)$ (from step 3 to 9). The job selection policy here tends to reuse the runtime environment between task packages and thus reduce resource management overhead.

4.2 Task Bundling

After node R_j is allocated to job $J_{i,k}$, we need further to decide which tasks of $J_{i,k}$ will be dispatched to R_j. Unlike the traditional HTC scheduling, we select a package of tasks rather than a single task in one dispatch. This task bundling approach is exactly the way to amortize the resource management overhead among tasks in the same package and increase the intra-node system utilization.

Our task selection approach considers the parallel degree $D_{i,j}$ of the application A_i on the node R_j, and chooses a subset of tasks of job $J_{i,k}$ by a bin-packing approach. The upper bound workload that each of $D_{i,j}$ bins can hold is determined by

$$\alpha = \max\{c_1 \times M_{i,k}, c_2 \times (T\_trans + T\_env_{i,j})\} \quad (9)$$

where $M_{i,k}$ is granularity of the largest task in job $J_{i,k}$, T_trans is the transmission startup overhead, $T\_env_{i,j}$ is the cost of runtime environment preparation for application A_i on R_j, and c_1 and c_2 are multipliers. Figure 3 shows the detail of the proposed scheme. The complexity is $O(s\log n_{i,k})$ (implemented by balanced Binary Search Tree), where s is the number of tasks in package. This is efficient enough for MTC applications even when there are millions of tasks.

The merits of this scheme are two-fold: 1) the workload of each job can be better balanced among different nodes by picking the largest task first; 2) the intra-node workload can be better balanced among processors as we have already simulated the intra-node scheduling by bin-packing.

Task-bundling ($J_{i,k}$, R_j)
1 Determine the bin upper bound workload α
2 Empty all $D_{i,j}$ bins: $B_h = \emptyset$
3 For each bin B_h
3.1 Find largest task $\tau \in J_{i,k}$, such that $|\tau| \leq \alpha - |B_h|$
3.2 If no such τ, complete the packing of B_h
3.3 Otherwise $J_{i,k} = J_{i,k} - \{\tau\}, B_h = B_h \cup \{\tau\}$; go to3.1
4 Bundle tasks: $B = \bigcup B_h$
5 Dispatch task package B to R_j

Fig. 3. Task Bundling Algorithm

5 Experiment

To study the proposed algorithm, we compare it with traditional heuristics, including MaxStd, MinMin and Sufferage. For MTC task scheduling, these heuristics can be used in two ways. The bare way is to schedule MTC tasks based on ETC matrix as in HTC. The other way is to revise them to fit into our scheduling framework. The revised heuristics run based on processing rate matrix rather than ETC. They play the role just as the first 3 steps of job-resource matching algorithm showed in Figure 2. The rest part of job-resource matching algorithm and the whole task-bundling algorithm will remain unchanged for revised heuristics.

5.1 Simulation method and parameters

The major parameters include node number m, application number p, job number n and task number for each job. And the generating methods for other items are as follows:

• Node. The core number of each node is uniformly sampled from {1, 2, 3, 4, 6, 8, 10, 12, 16, 32}.
• Application. An *IO-to-CPU ratio* is attached to each application, representing the application's preference for I/O or CPU speed. The ratio value is uniformly sampled from range (0,1). Large values indicate IO-intensive applications and small ones the CPU-intensive applications.
• Application's parallel degree matrix $D_{i,j}$ .The $D_{i,j}$ value is determined by $D_{i,j} = \lceil (1 - \gamma_i) \times cn_j \rceil$, where γ_i is IO-to-CPU ratio of application A_i, and cn_j is the core number of node R_j.
• Application's processing rate matrix $r_{i,j}$.We use the CVB method proposed in [15] to generate the application-resource processing rate matrix. In CVB method, μ_{task}, V_{task} and $V_{machine}$, which represent the mean of task execution time, CV of tasks and the CV of machines, are the three parameters to control the ETC matrix. In the context of application processing rate, V_{task} is actually $V_{application}$, indicating the heterogeneity level of application; $V_{machine}$ remains the same meaning; and μ_{task} becomes a scale parameter without any significance, and is assigned a fixed value 100. $V_{machine}$ and $V_{application}$ will be varied with values {0.1, 0.6}, representing low and high heterogeneity respectively.
• Job. Each job belongs to exactly one of the p applications, uniformly. The tasks' *actual task complexity* of each job is drawn from a Power Law distribution, with mean granularity 100 seconds and minimum granularity 1 second measured on the reference node.
• Task. Based on the *actual task complexity*, the *estimated task complexity* is sampled from a truncated Gaussian distribution $N(\text{actual\_complexity}, \theta \times \text{actual\_complexity})$ as in [9], where θ is a coefficient controlling the inaccuracy of prediction. The inaccuracy level is application-specific, and is uniformly drawn from range $[0.5,1.5]$.
• Resource management overhead. For application A_i, this value is $\text{Overhead}_i = 10 \times \sigma(\gamma_i)$,where σ is logistic function with 0.5 as location parameter and 0.1 as shape parameter, and 10 is a parameter controlling the maximal overhead.

Under the same parameters set, experiment will be repeated 10 times and the results will be averaged to eliminate casual effects.

5.2 Evaluation of the proposed algorithm

This experiment is to evaluate the application-level scheduling with task bundling algorithm (**ALSTB**) against the traditional heuristics comprehensively. Values of the major parameters are listed in Table 1. Value of task number per job here is set to be much smaller than real MTC cases, as the traditional task-level scheduling is too slow for larger values.

Figure 4(a) shows that when number of tasks is small, traditional heuristics behave pretty well; but as the task number increase, the proposed algorithm becomes better. This proved that the traditional heuristics cost more time in resource management and thus deteriorate the performance. Figure 4(b) shows a gradually bigger advantage of

ALSTB in terms of Average Job Makespan when increasing the number of tasks. The reason is, unlike ALSTB, the bare traditional heuristics treat tasks from different jobs indiscriminately, which in its extreme case may schedule tasks from different jobs alternately and result in bad Average Job Makespan. Figure 4(c) shows a clear margin between ALSTB and other three heuristics. We note that the bare heuristics schedule one task every time and cannot fully utilize a modern multi-core node, while the ALSTB considers the intra-node capacity exploitation and workload balancing by the bin-packing task-bundling strategy. This makes the significant difference of Overall Flowtime observed in graph.

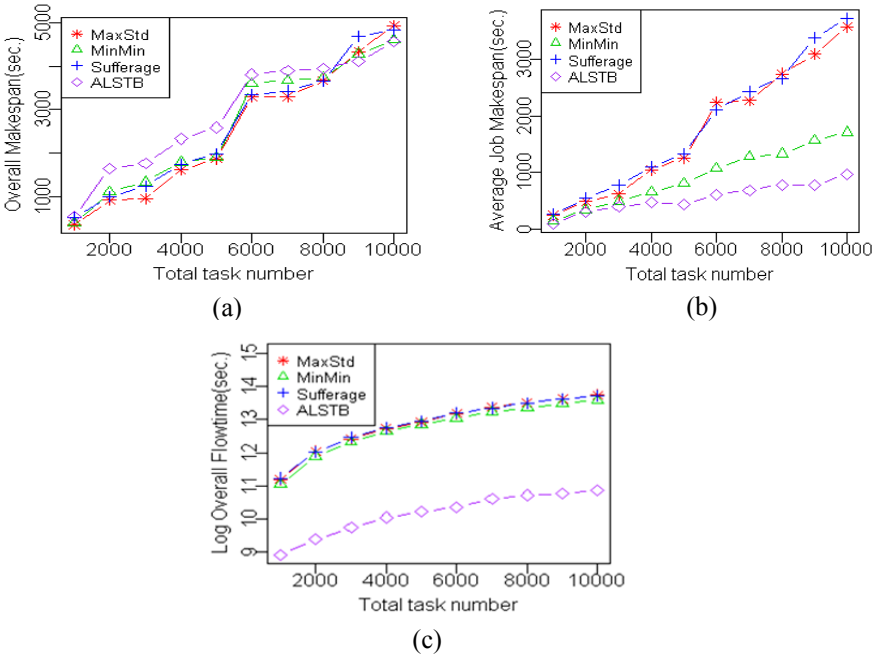


Fig. 4. Performance impact of task number on Overall Makespan, Average Job Makespan and Overall Flowtime

5.3 Evaluation of the revised heuristics

This experiment is to evaluate the revised heuristics (denoted as R-MaxStd, R-MinMin and R-Sufferage) that have been adapted to fit into our scheduling model. The experiment is put in real MTC conditions, as listed in Table 2. Every job's task number is uniformly between 10,000 and 50,000, and the total number of processed tasks will range from 1,000,000 to 5,000,000.

Table 1. Major parameters (I)

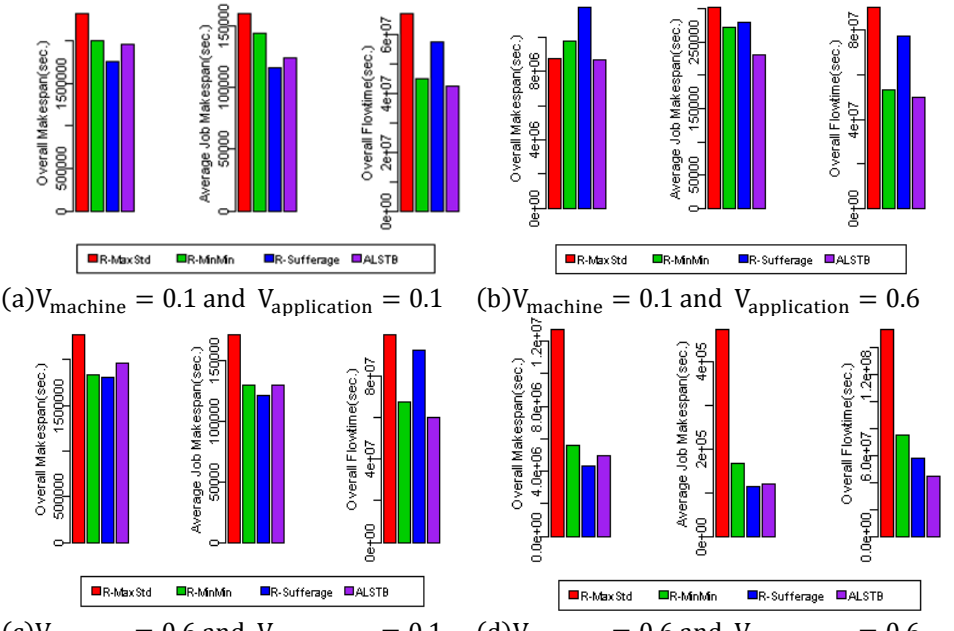
NodeNum	ApplicationNum	JobNum	TaskNumPerJob
100	10	20	50-500

Table 2. Major parameters (II)

NodeNum	ApplicationNum	JobNum	TaskNumPerJob
1,000	50	100	10,000-50,000

Figure 5(a) shows the results when both application and resoruce heterogeneity are low. R-Sufferage is the best in terms of Overall Makespan and Average Job Makespan, whereas ALSTB achieves the best Overall Flowtime.The reason shall be that ALSTB has taken both the application and resource's perspectives into consideration and tend to make a balanced scheduling decision. Figure 5(b) shows the results for high application heterogeneity and low resource heterogeneity, where ALSTB exhibits better than three other heuristics in all metrics. Figure 5(c) shows the results for low application heterogeneity and high resource heterogeneity, which is similar to Figure 5(a). Finally Figure 5(d) shows the results for high application heterogeneity and high resource heterogeneity. The algrithms' performance is more dispersed than other settings, especially that R-MaxStd is far worser than others.

We can read two things in these four results. First, compared to the bare heuristics, the performance of revised heuristics are greatly improved, which could even compete with ALSTB; this verifies the merit of the proposed scheduling strategies. Second, ALSTB still has its advantage in that it gives lower Flowtime in all four settings.



(a)$V_{machine} = 0.1$ and $V_{application} = 0.1$ (b)$V_{machine} = 0.1$ and $V_{application} = 0.6$

(c)$V_{machine} = 0.6$ and $V_{application} = 0.1$ (d)$V_{machine} = 0.6$ and $V_{application} = 0.6$

Fig. 5. Performance of the revised heuristic V.S. ALSTB in 4 settings

6 Conclusions

In this paper, we propose the ALSTB algorithm, which divides the whole scheduling into a job-resource matching algorithm and a bin-packing based task-bundling algorithm for multiple MTC applications scheduling in heterogeneous environments. The proposed algorithm is compared against both bare and revised traditional heuristics, where the latter are adapted from the former to fit into our scheduling framework. The simulation experiments show that our algorithm gets better results than the bare traditional heuristics, and outperforms the revised ones in Flowtime and meanwhile achieves performance in Makespan comparable to them. So the conclusions are two-fold: on one hand, ALSTB provides a way for multiple applications many-tasks scheduling, and on the other hand, ALSTB gives the balanced optimization of Makespan and Flowtime. In future, our research will concern the fairness issues involved in the multi-user Many-Task Computing.

References

1. Ioan Raicu, Zhao Zhang, Michael Wilde, Ian T. Foster, Peter H. Beckman, Kamil Iskra, Ben Clifford. Toward Loosely Coupled Programming on Petascale Systems, IEEE/ACM SuperComputing 2008.
2. I. Raicu, I. Foster, Y. Zhao. Many-Task Computing for Grids and Supercomputers, IEEE Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS08), 2008
3. Michael Isard, Vijayan Prabhakaran, Jon Currey,Udi Wieder, Kunal Talwar and Andrew Goldberg, Quincy: Fair Scheduling for distributed Computing Clusters, OSDI'2010
4. Namshin Kim, et al.ECgene: Genome-based EST clustering and gene modeling foralternative splicing. Genome Res. 2005, 15: 566-576.
5. Huashan Yu, Yingnan Li, Xianguo Wu, Jian Xiao, Xiaoming Li , A Self-Optimizing Computation Partitioning Algorithm for Distributed Many-task Computing, ChinaGrid' 2010
6. J. Xu, A.Y.S. Lam, and V.O.K. Li, Chemical reaction optimization for task scheduling in grid computing, IEEE Trans. Parallel Distrib. Systems, 2011
7. M. R. Garey and D. S. Johnson, 'Strong' NP-CompletenessResults: Motivation, Examples, and Implications, J. Associationfor Computing Machinery, vol. 25, no. 3, pp. 499-508, Jul. 1978.
8. T.D. Braun, D. Hensgen, R.F. Freund, H.J. Siegel, N. Beck, L.L. Boloni, M. Maheswaran, A.I. Reuther, J.P. Robertson, M.D. Theys, B. Yao, A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems, J. Parallel and Distributed Comput. 61 (6) (2001) 810–837.
9. M. Maheswaran, S. Ali, H.J. Siegel, D. Hensgen, R.F. Freund, Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems, in: Proceedings of the Eighth Heterogeneous Computing Workshop, 1999.
10. Munir, E.U., Jian-Zhong Li, Sheng-Fei Shi, Zhaonian Zou and Donghua Yang. MaxStd: A Task Scheduling Heuristic for Heterogeneous Computing Environment. Information Technology Journal, ISSN-1812-5638.
11. H. Izakian, B.T. Ladani, K. Zamanifar, and A. Abraham, A Novel Particle Swarm Optimization Approach for Grid Job Scheduling,Proc. Third Int'l Conf. Information Systems, Technology and Management,vol. 31, pp. 100-109, Mar. 2009.

12. A. Iosup, O. O. Sonmez, S. Anoep, and D. H. J. Epema. The performance of bags-of-tasks in large-scale distributed systems. International Symposium on High-Performance Distributed Computing (HPDC), pages 97–108. ACM, 2008
13. I. Raicu, et al. Falkon: a Fast and Light-weight tasK execution framework. IEEE/ACM SuperComputing 2007.
14. Yingnan Li, Xianguo Wu, Jian Xiao, Yu Zhang, Huashan Yu, A Scheduling Algorithm Based on Task Complexity Estimating for Many-Task Computing, SKG'2010
15. S. Ali, H.J. Siegel, M. Maheswaran, S. Ali, D. Hensgen, Task execution time modeling for heterogeneous computing systems, in: Proceedings of the Ninth Heterogeneous Computing Workshop, 2000, pp. 185–200.