# A Lightweight Execution Framework for Massive Independent Tasks

Li Hui
*Shenzhen Graduate School of Peking University*
*Shenzhen, Guangdong 518055, P.R.China*
*lihui@net.pku.edu.cn*

Yu Huashan
*Institute of Network Computing and Distributed System, Peking University*
*Beijing 100871 P.R.China*
*yuhs@pku.edu.cn*

Li Xiaoming
*State Key Laboratory of Advanced Optical Communication & Network, Peking University, Beijing 100871, P.R.China*
*lxm@pku.edu.cn*

## Abstract

*This paper presents a lightweight framework for executing many independent tasks efficiently on grids of heterogeneous computational nodes. It dynamically groups tasks of different granularities and dispatches the groups onto distributed computational resources concurrently. Three strategies have been devised to improve the efficiency of computation and resource utilization. One strategy is to pack up to thousands of tasks into one request. Another is to share the effort in resource discovery and allocation among requests by separating resource allocations from request submissions. The third strategy is to pack variable numbers of tasks into different requests, where the task number is a function of the destination resource's computability. This framework has been implemented in Gracie, a computational grid software platform developed by Peking University, and used for executing bioinformatics tasks. We describe its architecture, evaluate its strategies, and compare its performance with GRAM. Analyzing the experiment results, we found that Gracie outperforms GRAM significantly for execution of sets of small tasks, which is aligned with the intuitive advantage of our approaches built in Gracie.*

## 1. Introduction

In the bioinformatics application of alternative splicing [1], there are tens of thousands of independent tasks. Each one takes some files as input, and the output is one or more files. They are executed serially and the time costs vary from less than one second to more than one week. The character that a set of independent tasks are contained in the problem domain is typical for a wide range of scientific applications, like ACTOR [2], SAT [3], multi-objective optimization problems [4] and genetic algorithms [5], etc. As defined in [6], these applications are Many Tasks Computing (MTC). They need a High-Throughput Computing (HTC) system to meet the desired performance.

With the Grid technology, HTC systems can be

*This work is supported by 973 (2007CB310902）, 863 project (2006AA02Z334), CNTP (2005DKA64001), and Scientific Research fund for Public Welfare (GYHY200806018).

978-1-4244-2872-4/08/$25.00 ©2008 IEEE

built with distributed, heterogeneous computational resources on the Internet. Nowadays, most Grid systems aiming at HTC depend on GRAM to access remote resources, like Condor-G [7], GridWay [8]. When a batch job has been accepted, the Grid system selects an appropriate supercomputer and submits it to the GRAM service, which interacts with the supercomputer's Local Resource Manager (LRM) to implement the request automatically. By implementing a consistent interface for different LRMs like PBS [9] and Condor [10], GRAM plays an important role for these Grid systems. However, the GRAM-based strategy process each task with one request, and selects, allocates resources for different requests independently. The extra overhead introduced by these operations has significant effect on the performances of MTCs. Figure 1 is an experimental result to illustrate this overhead. The experiment was made on Peking University campus network. A SMP server with 4 cores was accessed remotely to perform 57 tasks in series. Comparing with transmitting the data files with Ftp and starting the task manually, GRAM consumed nearly 8 more seconds for every task.

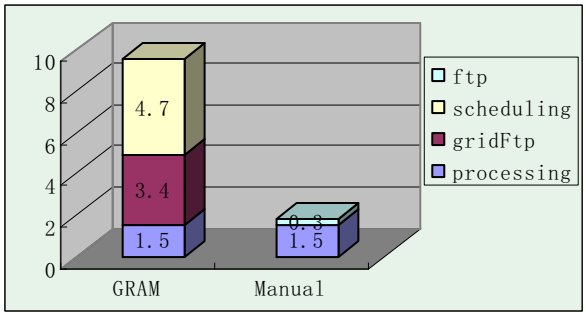


**Figure 1. Average cost per task profiling**

This paper focuses on improving the efficiency of MTC on Computational Grids. A lightweight execution framework has been proposed and three optimizing strategies have been devised. One strategy is to pack up to thousands of tasks into one request, so as to improve remote resource accessing efficiency. Another strategy is to separate Grid requests with resource allocations, so as to share resource discovering and allocation among requests. The third strategy is to partition a MTC application's task-set dynamically and non-uniformly, different subsets are performed concurrently on distributed resources, so as to do parallel computing on the Grid and balance task workload among distributed resources. The framework provides a set of APIs for MTC applications to submit task-sets, and process the submitted tasks with Grid resources automatically and efficiently.

Section 2 discusses related works. Section 3 illustrates the framework's main components and optimizing strategies. Section 4 introduces its implementation, and a bioinformatics computational application developed with this implementation is also introduced. Section 5 presents the experimental results to evaluate the framework and its optimizing strategies.

## 2. Related work

GRAM-based meta-scheduler such as Condor-G, GridWay is able to schedule different tasks to distributed resources. The overhead for extra operations like resource allocating and accessing is significant for MTC, and may counteract the effect of parallel computing in some cases. Falkon [11] made an attempt to reduce average scheduling overhead of one task on clusters. It uses multi-level scheduling strategy to separate resource acquisition from tasks dispatch. It uses LRM (via GRAM) to activate executor (daemon process deployed on compute node). After that it bypasses LRMs and dispatches tasks directly to executors. It also employs the task bundling strategy [11,12] to decrease communication cost for task. These strategies help Falkon decrease scheduling cost per task caused by using GRAM and LRMs. The experiments of running 'sleep 0' tasks indicate that its throughput increases from 604 tasks/sec, without grouping, to a peak of 3773 tasks/sec with grouping. However, it does not provide further information about its performance on Grid system consists of distributed workstations/PCs.

## 3. Architecture

The lightweight execution framework proposed in this paper aims to efficiently execute massive independent tasks on computational Grids. We assume that the tasks are independent, which means they have no interaction with each other and no workflow is required to denote their executing order. We assume that the task-sets are static. In another word, all tasks have been specified before a MTC is executed. We also assume that all application software required by the tasks have been previously installed on compute nodes that are possibly employed for the MTC.

### 3.1. Architecture

The lightweight execution framework consists of grid APIs, resource manager, and adaptive job processor, as illustrated in figure 2. Grid APIs are a set of C++ class for MTC applications to process task-sets transparently with remote resources. Resource manager is a grid service to allocate and monitor remote resources for MTC applications. Adaptive job processor takes in charge of executing task and transferring input/output data on remote resources.

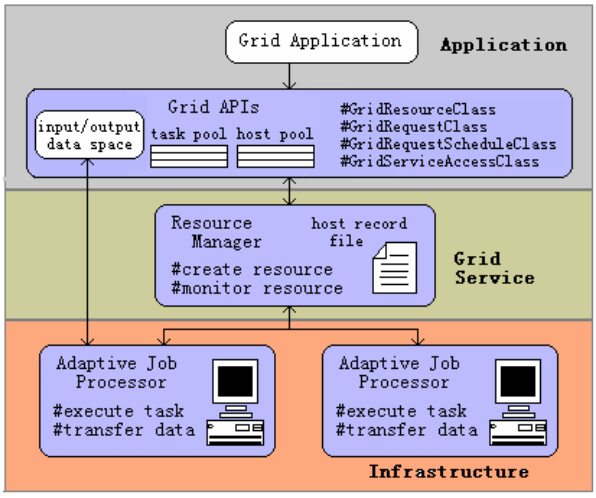


**Figure 2. Framework architecture**

**3.1.1 Grid APIs.** Grid APIs aims to enable application programmer to develop Grid application easily. Moreover, it also aims to execute massive independent tasks in Grid environment efficiently. To achieve these two objectives, four important classes are designed:

-GridResourceClass
-GridRequstClass
-GridRequestScheduleClass
-GridServiceAccessClass

Every task execution environment on remote host is represented with one instance of GridResourceClass, which is responsible for the details of submitting and executing tasks on the execution environment. The instance is created and owned by a MTC application itself instead of its tasks. Application programmer can determine the time to allocate and release resources.

GridRequestClass manages a task pool containing all tasks of a MTC application. It dynamically groups up to thousands of tasks into one grid request. For different request, the number of grouped tasks is varied, depending on computability of the resource which the request is submitted to.

GridRequestScheduleClass schedules grid requests to grid resource (instance of GridResourceClass). It implements several scheduling strategies to improve resource utilization, including: OLB [13] and balanced task workload scheduling strategy. OLB assigns a resource with a new grid request immediately after its status changes to be idle. The latter strategy dynamically tunes each request's workload, and makes the grid request's workload to match the computing power of selected resource.

GridServiceAccessClass implements the details of accessing resource manager for creating, releasing, monitoring grid resources. It is the basis of other three classes.

**3.1.2. Resource manager.** Resource manager is a web service conforming to the WSRF specification. It encapsulates heterogeneous resources to be WS resources with consistent interfaces. By implementing the factory/instance pattern [14], it allows creating and

managing multiple resources concurrently. For each resource creating requirement, it selects one appropriate machine resource and activates the adaptive job processor remotely.

Resource manager also takes the responsibility of monitoring resources status. Every resource contains properties to record its own status such as "BUSY" or "FREE" during computation. Grid application can decide whether to dispatch a new request to that resource based on its status information.

**3.1.3. The Adaptive Job Processor (AJP).** The AJP is deployed on compute node in advance. Its functions include: scheduling tasks on compute host; managing input/output data staging among submit host and compute host.

AJP executes multiple tasks concurrently to exploiting the parallel computability of multi-core or multiprocessor machines. Besides, AJP takes the responsibility of transmitting data files for Grid requests. It downloads input data files before executing a grid request's tasks, and uploads output data files after all tasks of the request has been completed.

## 3.2. Strategies

To improve the efficiency of MTC, we have devised and implemented a set of strategies for the lightweight execution framework, which includes task grouping, user-oriented resource allocating [15], and balanced task workload scheduling.

**3.2.1 Task grouping strategy.** Grid application can dynamically group up to thousands of tasks within one grid request, whose size is determined by several strategies illustrated later. One objective is to decrease the number of requests, which in turn decrease the cost of submitting requests and starting up of remote data transmission. Another objective is to provide AJP with enough independent tasks to utilize the parallel computing power of multiprocessor machines.

**3.2.2 User-Oriented resource allocating strategy.** The lightweight execution framework allocates resources to grid application rather than any requests or jobs. This strategy separates resource allocation from task dispatching, allowing one resource to execute as many requests as desired in its lifetime. With this strategy, a MTC can significantly reduce its cost for resources allocating, configuring and releasing.

**3.2.3. Balanced task workload scheduling strategy.** The balanced task workload scheduling strategy aims to balance request workload among machines with different computing power. Grid request is dynamically created with task grouping strategy, and its workload is calculated by an experience mathematic formula, so as to match the computing power of selected resource.

The appropriate amount of workload $w_i$ for resource $r_i$ in a round is defined as

$$w_i = \partial \cdot n_i \cdot b_w + \frac{(1-\partial) \cdot T_w \cdot n_i}{2 \cdot \sum_{i=1}^{m} n_i}$$

Where $n_i$ refers to the amount of processors of resource $r_i$, $\sum_{i=1}^{m} n_i$ refers to the sum of processors of all resources allocated to the MTC. $b_w$ is the basic workload dispatched to one CPU, and pre-defined for each MTC from experience. For instance, for data-intensive application, it is the data size of task. For compute-intensive application, it is estimated CPU time of task. $T_w$ refers to the sum of workload of all tasks. $b_w$ and $T_w$ have the same unit. $\frac{T_w \cdot n_i}{\sum_{i=1}^{m} n_i}$ refers to the amount of workload assigned to resource $r_i$, called its responsible workload. The responsible

workload of resource $r_i$ is divided by 2, so as to avoid too big workload dispatched to a resource in a round. The factor $\partial$ is a real ranging from 0 to 1. The factors $\partial$ and $(1-\partial)$ are tuned to a balance between basic workload and half of responsible workload of resource $r_i$. Increasing the value $\partial$ will decrease the value $w_i$, to the contrary, will increase the value $w_i$.

In different applications, the factors $\partial$, $b_w$ and $T_w$ will have different value, unit, and semantics. GridRequestScheduleClass provides interface that enables programmer to assign these parameter by experience.

With this strategy, different machine resources are likely to complete their requests in the same time. Accordingly, a MTC's time cost is minimized.

## 4. Implementation and application

### 4.1. Gracie platform

The lightweight execution framework has been implemented in Gracie, a computational grid software platform developed by Peking University. It consists of Pier, Fasee, Madie, User-oriented Resource Manager and Adaptive Job Processor, as illustrated in figure 3. Pier provides an API for managing Grid resources, accessing and processing tasks with these resources in C/C++ applications. Fasee implements the capability of discovering and selecting resources in computational Grids. The user-oriented resource manager service encapsulates heterogeneous resources to be WS resources with consistent interfaces, and is responsible for creating and managing these WS resources. Madie provides Grid applications with capability to monitor and retrieve a set of WS resources that may be provided by different user-oriented resource manager services.

Fasee, Madie and the user-oriented resource manager service are WSRF services developed with Globus Toolkit 4. Pier is implemented with C++, and JNI technology is used to interact with these underlying Web services. And the AJP is also implemented with C++.

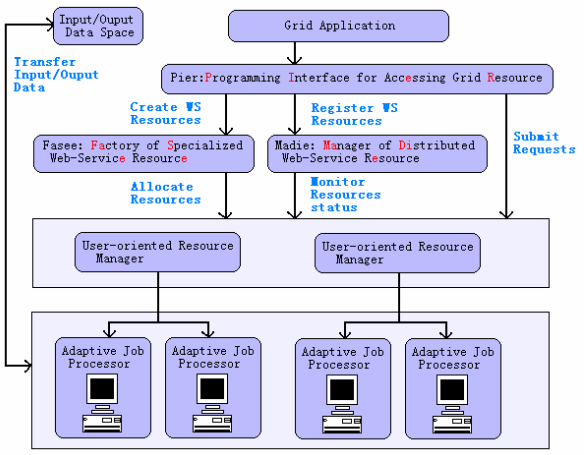


**Figure 3. Gracie architecture**

Every computer in the Grid has been installed the AJP separately and is registered to some user-oriented resource manager service. The service and the AJP communicate through Telnet protocol. Several distributed computers are allowed to be registered to one user-oriented resource manager service. And Ftp protocol is used by AJP to transmit data files between submission host and computing host.

Grid applications are developed in C++, using the Grid API defined by Pier. Programmers use C++ to implement the application logic and algorithm, and use the Grid API to allocate and retrieve Grid resources, pack tasks to be Grid requests, and submit Grid requests to distributed resources, and monitor the progress of submitted Grid requests. Different requests are submitted to distributed resources to be processed concurrently.

### 4.2. Application

We have developed a Grid application gSVAP with Gracie for CBI (China Biology Information Center) at Peking University to perform genome alterative splicing (AS). The object is to utilize distributed SMP workstations to improve computing performance by performing AS computations of different genes on different workstations concurrently.

**4.2.1. Alternative splicing application.** A gene contains numerous exons and introns, and the exons can be spliced together in different ways. For example, if a gene contains 10 exons, one version of the mRNA transcribed from that gene might contain exons 1-9. Another version of the mRNA might contain exons 1-8, and exon 10. This is called alternative splicing (AS), and can produce different forms of a protein from the same gene. Detection of AS relies heavily on high-throughput sequencing of expressed sequence tag (EST) sequences, involving complex computation of massive expression data. The application is to predicate all mRNAs that are possible transcribed from a genome.

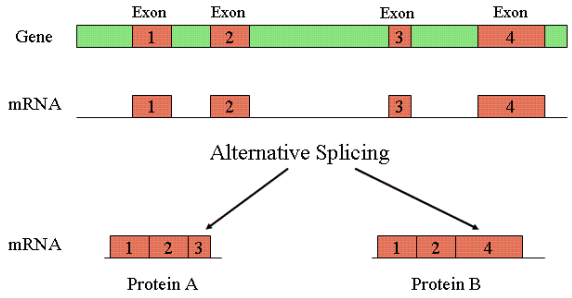


**Figure 4. Process of alternative splicing**

http://www.ncbi.nlm.nih.gov/Class/MLACourse/Modules/MolBioReview/alternative_splicing.html

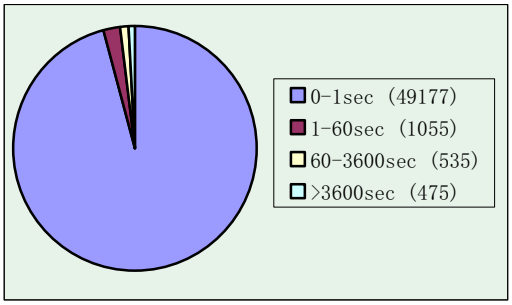


**Figure 5. Execution time distribution**

A genome consists of a large amount of genes. The complexity of gene structures are varied greatly, leading to the computing overhead is irregularly distributed among different genes. The analysis process for one gene may last less than 1 second, while it may last more than one week for some others. For each gene, the analysis process is defined in the AS application to be one task and is to be executed on the Grid. The ESTs belongs to one gene are saved as input file of one task. Figure 5 illustrates the application's irregularity. There are totally 51242 tasks in the problem domain, about 98.0% of them cost less than 60 seconds.

## 5. Performance and strategies evaluation

To evaluate the strategies discussed above and Gracie's performance, we have made some experiments with the AS application. 10 SMPs machines were utilized by Gracie to perform tasks for the AS application, and they were distributed on Peking University campus network. AS application was running on SMP_1. The experiment input data was human genome's ESTs published by Gen Bank in 2008 Jan, ESTs belonging to different genes were saved in different files. Those EST files were also stored on machine SMP_1. The User-oriented resource manager service, Fasee and Madie were deployed on SMP_2. The adaptive job processor and the gene AS software tool have been previously installed on the other 8 SMPs, which were managed by one User-oriented

resource manager service deployed on SMP_2. In the experiments, we assigned every task an up bound for the execution time. A task was terminated by the adaptive job processor when the execution time exceeded the up bound.

### 5.1. Evaluation metrics

The two main metrics in the experiments are resource utilization and execution efficiency:

$$res\_utilization = \frac{\text{Ttransfer} + \text{Tprocess}}{\text{Tidle+ Ttransfer+ Tgrid+ Tprocess+Tsys}}$$

$$exec\_efficiency = \frac{\text{Ttransfer} + \text{Tprocess}}{\text{Tbind+ Ttransfer+ Tprocess+Tsys}}$$

$T_{transfer}$ refers to the time used to transfer input/output data. $T_{process}$ refers to the time used to execute tasks. $T_{sys}$ refers to the time used to schedule tasks to resources. $T_{idle}$ refers to the duration when the resource has been allocated to grid application, but is idle. $T_{grid}$ refers to the time used to run grid software. $T_{bind}$ refers to the time used to create and release resources.

### 5.2. Strategy evaluations

**5.2.1 User-Oriented resource allocating strategy.** This experiment was to make a comparison between the request-oriented resource allocating strategy and our user-oriented resource allocating strategy. 10 tasks were executed on one SMP, and the task grouping strategy was deactivated in the experiment.

Figure 6 is the experimental result. When the request-oriented strategy was used, the cost of resources creating, registering, and releasing of 10 tasks are 13.37, 2.2, and 2.5 seconds respectively. When the user-oriented strategy was used, the costs of those operations of are 1.6, 0.23, and 0.206 seconds respectively. Comparing with the request-oriented strategy, more than 22 seconds were saved by the user-oriented strategy.

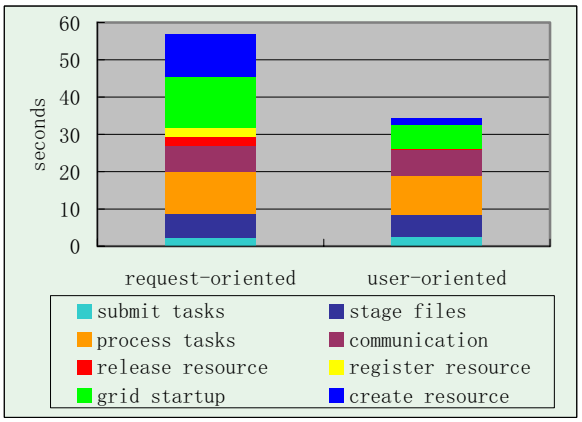


**Figure 6. Ten tasks cost profiling for two strategies**

**5.2.2. Task grouping strategy.** This experiment was to evaluate the task grouping strategy's effect on remote resource accessing efficiency. All 51242 tasks were submitted to one SMP workstation, which performed computation for each submitted task. The experiment was repeated 20 times, each time a different $w_i$ was specified to the request size. In the AS application, $w_i$ is total size of a request's input files.

Figure 7 illustrates the experiment result. The time required to submit all 51242 tasks were reduced from 1688 seconds to minimum 1008 seconds, when $w_i$ was increased from 0.5MB to 100MB. And its throughput increased from 30.3 tasks/sec to 50.7 tasks/sec.

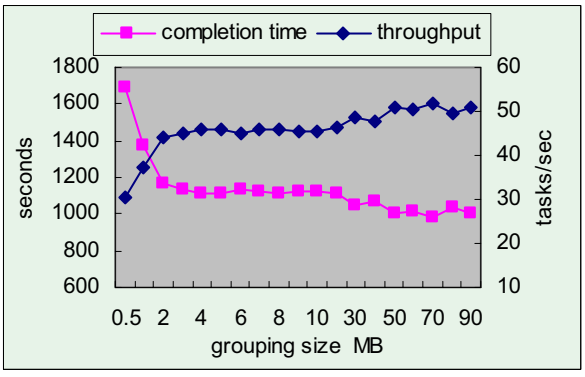


**Figure 7. Completion time and throughput with various grouping size**

**5.2.3. Balanced task workload scheduling strategy.** We made a comparison between our balanced task

workload scheduling strategy and the fixed workload scheduling strategy. In the latter strategy, the request size is a constant for different resources. This experiment exploited 6 SMP workstations to execute all 51242 tasks, and each one's maximum execution time was set to be 60 seconds. In the fixed workload scheduling strategy, the task grouping size was set to be 6.5MB, which was selected from a set of candidates to deliver the best performance. In the balanced task workload scheduling strategy, its $\partial$ value was set to be 0.3.

To process all 51242 tasks, our balanced task workload scheduling strategy cost 3219 seconds, while the fixed workload scheduling strategy cost 3703 seconds. Figure 8 illustrate the work time (task processing time + file staging time) distribution on 6 SMPs for two strategies. It is indicated that the work time distribution of balanced strategy is more even than that of fixed strategy.

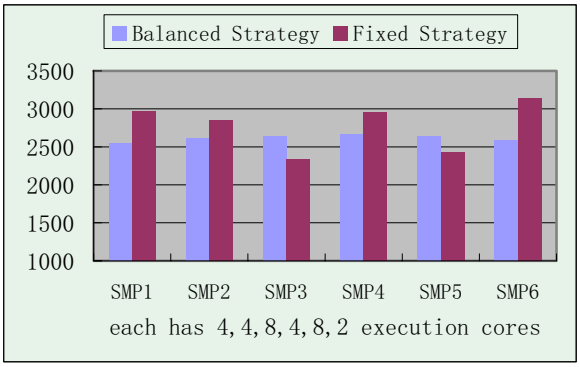


**Figure 8. Work time distribution on 6SMPs for two strategies**

### 5.3. Gracie vs. GRAM

The experiment was to evaluate resource utilization in Gracie by comparing with GRAM. We devised 4 experiments to submit tasks with Gracie and GRAM respectively: GRAM, GRAM+Fork, Gracie, and Gracie+Optimize. The GRAM experiment was to submit tasks with GRAM, and a new task would not be submitted until the last submitted one has been completed. The GRAM+Fork experiment utilized 4 concurrent threads to submit tasks, and each thread performed just the same operations as in the GRAM experiment. The Gracie experiment submitted tasks with Gracie, and each request contained 4 tasks. In the Gracie+Optimize experiment, all optimizing strategies discussed above were used.

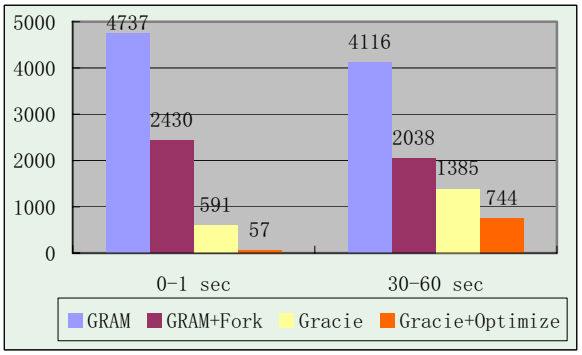


**Figure 9. Comparison of four approaches**

We selected two set of tasks. One set consists of 1139 tasks whose time costs are less than 1 second. Another set consists of 86 tasks whose time costs are ranged from 30 seconds to 60 seconds. Figure 9 is the experimental result. It indicates that Gracie uses less 87% and 67% completion time than GRAM does in the two experiments respectively.

## 6. Summary and future work

### 6.1. Summary

In this paper, we introduced a lightweight execution framework for scheduling massive independent tasks on distributed computational resources. To make the execution efficiently, it uses tasks grouping strategy and user-oriented resource allocating strategy to reduce average scheduling cost of one task. Further, it uses a balanced task workload scheduling strategy to minimize the completion time of whole tasks. Experiments in this paper indicate that the proposed framework can accelerate the execution of massive independent tasks in Grid environment. What is more, it allows application programmers to easily develop

Grid application with a set of Grid APIs.

### 6.2. Future work

With multi-core architecture becomes popular, we plan to make the system adaptive to different architectures of computational nodes in a cluster in next version. Gracie not only aims to exploit the multi-node parallelism of cluster but also aims to exploit the parallelism in multi-core of each node of a cluster. By employing the two levels of parallelism, we expect Gracie will deliver even higher performance than GRAM.

### 6.3. Acknowledgements

Other teammates from EECS of Peking University have made a great contribution to the Gracie system. They are Wang Yang, Du Zhongxuan who made most of the code of Gracie, and testing. We also thank the members of CBI who inspired this project and help us to learn about the basic knowledge of bioinformatics.

## 7. Reference

[1] Namshin Kim, et al., ECgene: Genome-based EST clustering and gene modeling for alternative splicing, Genome Res. 2005 15: 566-576.

[2] J. Brazile, et al., Cluster vs. Grid for Operational Generation of ATCOR's MODTRAN-based Look Up Tables, Parallel Computing (2007), doi: 10.1016/j.parco.2007.11.002.

[3] W. Chrabakh, et al., GridSAT: a system for solving satisfiability problems using a computational grid, Parallel Computing 32 (2006) 660–687

[4] F. Luna et al., Observations in using Grid-enabled technologies for solving multi-objective optimization problems, Parallel Computing 32 (2006) 377–393.

[5] D. Lim et al., Efficient Hierarchical Parallel Genetic Algorithms using Grid computing, Future Generation Computer Systems 23 (2007) 658–670

[6] Ioan Raicu, Zhao Zhang, Mike Wilde, Ian Foster, Pete Beckman, Kamil Iskra, Ben Clifford. "Towards Loosely-Coupled Programming on Petascale Systems", to appear at IEEE/ACM Supercomputing 2008.

[7] J. Frey, T. Tannenbaum, I. Foster, M. Frey, S. Tuecke, "Condor-G: A Computation Management Agent for Multi-institutional Grids," Cluster Computing, 2002.

[8] GridWay System www.gridway.org

[9] B. Bode, D.M. Halstead, R. Kendall, Z. Lei, W. Hall, D. Jackson. "The Portable Batch Scheduler and the Maui Scheduler on Linux Clusters", Usenix, 4th Annual Linux Showcase & Conference, 2000.

[10] D. Thain, T. Tannenbaum, and M. Livny, "Distributed Computing in Practice: The Condor Experience" Concurrency and Computation: Practice and Experience, Vol. 17, No. 2-4, pages 323-356, February-April, 2005.

[11] Ioan Raicu, Yong Zhao, Catalin Dumitrescu, Ian Foster, Mike Wilde. "Falkon: a Fast and Light-weight tasK executiON framework", IEEE/ACM SC, 2007.

[12] Nithiapidary Muthuvelu, Junyang Liu, Nay Lin Soe, Srikumar Venugopal, Anthony Sulistio and Rajkumar Buyya1. "A Dynamic Job Grouping-Based Scheduling for Deploying Applications with Fine-Grained Tasks on Global Grids" Australasian Workshop on Grid Computing and e-Research (AusGrid2005)

[13] Braun TD, Siegel HJ, Beck N. A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems [J]. Journal of Parallel and Distributed Computing, 2001, 61 (1): 810 - 837.

[14] Ian Foster, "Globus Toolkit Version 4: Software for Service-Oriented Systems", IFIP International Conference on Network and Parallel Computing (NPC 2005). LNCS 3779, pp. 2 – 13.

[15] YU Huashan, Kong Lei, "Practices of Grid Computing for Genome Alternative Splicing Analysis" IEEE GCC 2007