A Multi-granular Application Management and Parallel Scheduling Model

Wei-Hua Bai[1,2]
1. School of Computer Science and Engineering
South China University of Technology
GuangZhou, China
2. School of Computer Science
ZhaoQing University
ZhaoQing, China
e-mail:bandwer@163.com

Jian-Qing Xi
School of Computer Science and Engineering
South China University of Technology
GuangZhou, China

Jia-Xian Zhu
School of Computer Science
ZhaoQing University
ZhaoQing, China

Shao-Wei Huang
School of Computer Science
ZhaoQing University
ZhaoQing, China

***Abstract*—We propose a novel multi-granular application management platform between PaaS and IaaS layers by using application virtualization techniques. We also investigate a parallel scheduling model on the platform. With the help of fine-grained application units, new functions can be created by combining the application units in different granularities based on business requirements. Moreover, we develop a multi-core-aware parallel scheduling model to process application requests, which not only increases the system flexibility and applicability, but also improves fine-grained computing resource allocation, the resource utilization of the fundamental infrastructure and system throughput.**

Keywords—Application Service; Web Service; Multi-core-aware; Parallel Scheduling

I. INTRODUCTION

As the development of applications in various areas, big data applications and the requirements of application services, as well as the design and development of the applications, have been increasingly complex and diverse. More IT resources are deployed in enterprise infrastructures (the IT infrastructure consists of hardware and software resources). At present, PaaS platforms provide users with the fundamental services of software development and research in the SaaS manner, in order to satisfy their changing business requirements[1,2,3,4,5]. Under this situation, although PaaS applications can be used to create enterprise applications in SOA architectures[6,7,8], there are drawbacks in building business systems on PaaS platforms[9]: (1) The requirement that integrates diverse types of business into an enterprise management system cannot be satisfied by the PaaS platforms; (2) The existing IT infrastructures in enterprises cannot be fully protected or utilized; (3) The task schedulers cannot fully utilized CPU resources based on the characteristics of applications and CPU architectures.

II. A MULTI-GRANULAR APPLICATION MANAGEMENT PLATFORM

A. The Application Management Platform

The structure of the application management platform is displayed in Fig.1. The platform integrates data in original enterprise systems into new business systems through web services, and implements the business logic by combining the web services in SOA mode. Moreover, the platform is encapsulated as independent business applications and exposes development interfaces to the third party or business developers. All data is stored in enterprise IT infrastructures. On the platform, applications are scheduled by a parallel scheduling engine. It is named AaaS (Application as a Service).

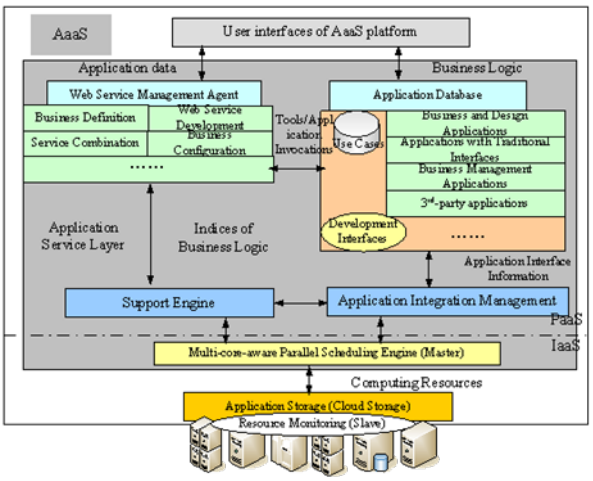

Fig. 1. Structure ofMulti-granular Application Management Platform

B. The Multi-granular Application Model

On the AssS platform, all applications are defined to be templates that can be generated by combining finer-grained applications. Two types of applications are provided by the system. One type is the standard applications that include database operations (database connection pool management, queries, updates, insertions and modifications), file operations, inputs/outputs and software interfaces; the other type is customized applications added by users based on their development patterns and specifications. All applications in the AaaS platform can be described by the following model.All the logic structures of application flows can be easily described by and stored in XML files.

Definition 1: Application Units (*Apps*): the smallest units of applications in all types and granularities, i.e. the finest-grained applications. An application unit is an executable application with independent inputs and outputs on an AaaS platform. A*Apps* can be abstracted by a quintuple:

Apps ={App_Info, App_DInfo, App_Input, App_Output, App_DSP}

Definition 2: Application Flow (*AppF*) is an application group that is comprised of *n (n≧1)* application units (*Apps*) or application flows (*AppF*) in a particular order. From the definition, an application flow is allowed to recursively combine application flows. An application flow can be represented by a tuple:

AppF = <Vapp, Era>

where, $Vapp=\{Vapp_i \mid Vapp_i \in Setof(Apps, AppF),\ 1 \leqq i \leqq n \text{ or } i=b \text{ or } i=e\}$ denotes an independent application set (called a set of vertices). $Vapp_b$ and $Vapp_e$ are two virtual applications, which represent the beginning and the end points in the application flow;

$Era=\{<Vapp_i, Vapp_j> \mid Vapp_i, Vapp_j \in Setof(Apps, AppF),\ 1 \leqq i \leqq n \text{ or } i=b,\ 1 \leqq j \leqq n \text{ or } j=e,\ i \neq j\}$ describes a pair of applications, indicating the relation of the caller and callee.

Definition 3: The granularities of application flows can be classified into three categories by the complexity of the application flows. The fine-grained application flows (*SG_AppF*), medium-grained application flows (*MG_AppF*), and coarse-grained application flows (*BG_AppF*) are defined as follows:

(1) fine-grained application flows (*SG_AppF*) are the ones that are made up of single application unit (*Apps*), *i.e.* $Vapp=\{Vapp_b, Vapp_1, Vapp_e\}$, $Era=\{<Vapp_b, Vapp_1>, <Vapp_1, Vapp_e>\}$, as shown in Fig.2(a).

(2) medium-grained application flows (*MG_AppF*) are the ones that consist of *n (n>1)* application units (*AppF*), *i.e.* $Vapp=\{Vapp_b, Vapp_1, Vapp_2, \ldots\ldots Vapp_n, Vapp_e\}$, $Era=\{<Vapp_b, Vapp_i>, \ldots\ldots <Vappj, Vappe>\}$, as shown in Fig.2(b).

(3) coarse-grained application flows (*BG_AppF*) are complex application flows that are generated by recursively combining application units (*Apps*) and application flows (*AppF*), as shown in Fig.2(c).

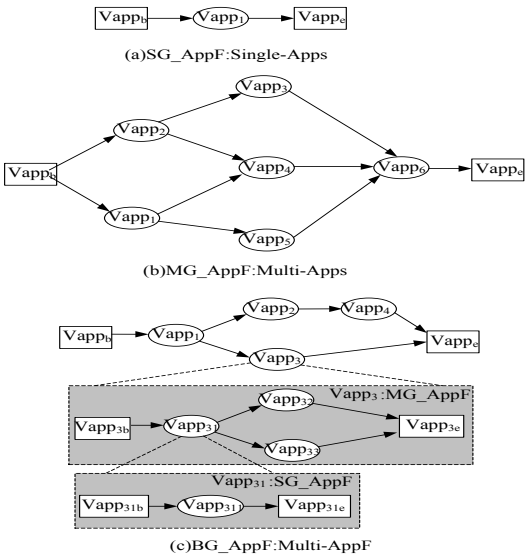

Fig. 2. Multi-granular Application flows

III. The Parallel Scheduling Model

On the AaaS platform, we design a two-level multi-core-aware parallel scheduling model that includes (1) a workflow scheduler in the master node and (2) a task scheduler in each computing node.

The structure of the multi-core-aware parallel scheduling model is displayed in Fig.3.

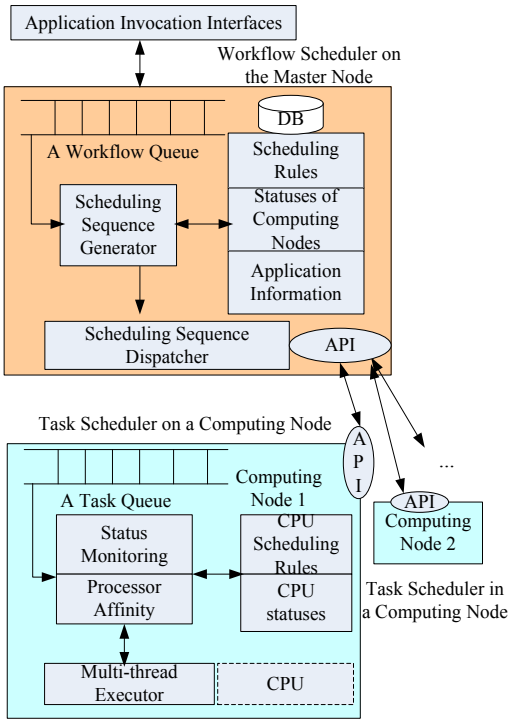

Fig. 3. structure of the multi-core-aware parallel scheduling model

(1) The workflow scheduler in the master node inserts application invocation requests into the workflow queue and

then generates the scheduling configuration XML files through the scheduling sequence generator (XML Code Generator) based on the pre-defined "scheduling rules", "statuses of computing nodes" and the application information (including the information of last execution, the application type and the execution time). Last, the workflow scheduler assigns the applications to corresponding computing nodes through the scheduling sequence dispatcher and sends responses to users.

Users can specify the scheduling rules to control the scheduling of application invocation requests. If no scheduling rule is set, a First-Come-First-Serve (FCFS) rule is applied as default, which assigns applications to idle computing nodes in order to keep the workload balance among the nodes.

The statuses of computing nodes include the architecture information of CPUs and the utilization of computing resources (CPU, memory, storage and network) on the node. These information is collected by status monitors on computing nodes and reported to the master node through heartbeat messages. The heartbeat messages also indicate that the computing nodes are in active statuses.

The application information module maintains the performance characteristics of applications. Based on these information, applications are classified into categories, and assigned to the computing nodes, the characteristics of which are similar with the performance characteristics of the applications in order to increase the system throughput.

(2) The task schedulers on computer nodes receive scheduling configuration XML files and application packages (if the applications are not deployed on the computing nodes) from the scheduling sequence dispatcher on the master node and insert the tasks into task queues. The default FCFS scheduling rule will be applied if no customized rule is specified. Then, the corresponding threads of the applications are bounded to particular CPU cores by using Processor Affinity techniques in order to achieve higher performance.

The status monitor records the thread statuses of applications on CPUs, which include CPU cycles, Cache miss rate, the execution time of the thread, and the number of instructions completed. With the help of these information, our module can obtain the execution characteristics of the applications and change the scheduling rules (policies) based on the feedback of the application execution if necessary.

The multi-thread executor uses Processor Affinity techniques, which bounds threads to specified CPU cores, assigns threads to idle CPUs or CPU cores on alternative chips. The executor is able to increase the CPU utilization by reducing the number of application migration among CPU cores, the waiting time and the Cache miss rate.

IV. EXPERIMENTS AND DISCUSSIONS

Hardware：① 2×Dell PowerEdge T720 (2×CPUs: Xeon 6-core E5-2630 2.3G, 12 cores in total); ② 1×Inspur NF8560M2 (4×CPU: Intel Xeon 6-core E7-4807 1.86G, 24 cores in total); ③ 3×LenovoM4336 (1×CPU: Intel 4-Core i7-3770 3.4G), 12 cores in total。

Software: OS-Linux kernel 2.6.21; AaaS Platform-JAVA1.6, JAVA RMI, JAVA Jna; Web-Tomcat6.5, Apache.

The objectives of experiments are to verify: ① the feasibility of the multi-granular application on the AaaS platform; ② the feasibility and effectiveness of the multi-core-aware parallel scheduling model; ③ the feasibility and effectiveness of the feedback mechanism in the parallel scheduling optimization.

Test bed setting: one Lenovo M4336 is used as the master node and a computing node; other servers are configured as computing nodes. In our experiments, we use three clients to simulate application requests and submit them to the master node. The application requests include:

(1) $Super\ \pi = 4\sum_{k=0}^{n}\frac{(-1)^k}{2k+1}, n = 10^7$, it is an*SG_AppF* application with CPU-intensive *Apps* by definition 3. Thus, the *AppF* type is called A.

(2)The calculation of$Super\ \pi$ ($n=10^8$) is divided into *M* (*M <= the number of CPU cores in the system,M* is set to be 6 in our experiments) tasks, in which *(M-1)* tasks calculate $(-1)^k/(2k+1), (k = 1,2,\dots,n)$ in parallel and the *M*th task calculates the sum of the intermediate results. By definition 3, this is an CPU-intensive application flow in *MG_AppF* type. Thus, the *AppF* type is called B.

(3)*WordCount* is an application that counts the number of words in files. In our experiments, we use a text file (the file size is 128 MBytes) and the output of *WordCount* is written to a specified file. By definition 3, *WordCount* is an I/O-intensive application in *BG_AppF* type. Thus, the *AppF* type is called C.

(4)We calculate the multiplication of two matrices of order *N* ($N=10^3$). By definition 3, the matrix multiplication is a CPU-intensive and memory dependent application in *MG_AppF* type. Thus, the *AppF* type is called D.

(5)We calculate *Kmeans (K=10)* of given *n* ($n=10^4$) points in a two-dimensional space, and the result is written to an external file. By definition 3, *Kmeans* algorithm is a CPU-intensive, I/O-intensive and memory dependent application in *BG_AppF* type. Thus, the *AppF* type is called E.

During the experiments, we use the following six configurations: ①1×M4336 (1 node, 4 cores); ②2×M4336 (2 nodes, 8 cores);③3×M4336(3 nodes, 12 cores); ④3×M4336 and 1×T720 (4 nodes, 24 cores); ⑤3×M4336 and 2×T720 (5 nodes, 36 cores); ⑥3×M4336, 2×T720 and 1×NF8560M2 (6 nodes, 60 cores). From the results of the experiments, we observe that:

(1)The execution time of applications in type C (I/O-intensive and memory dependent) is the greatest among that of all applications in our experiments, which is a major factor that the testing takes a long time.

(2)Based on the feedback information of applications and the rule of "setting a threshold on the number of threads in each core and setting a threshold on the number of threads in each type", our parallel scheduling model can avoid the performance degradation by assigning applications in the same type to

multiple computing nodes. The parallel scheduling model achieves more than 35% performance improvement in all other cases. The performance improvement becomes increasingly significant (over 80%) as the number of applications grows. Take an experiment for example, 8 applications in type C were waiting on a computing node, and the system took 536 seconds to complete these applications. If these applications can be assigned to two computing nodes in a balanced manner, they could be completed in 47 seconds, only one eleventh of the execution time in the previous case.

(3)A maximum value is set as a threshold to limit the number of threads on a node. The value of the threshold depends on the number of CPU cores on the node. Before reaching the threshold, the system performance linearly increases with the number of threads. But if the number of threads exceeds the threshold, the system performance nearly keeps unchanged. Even when the number of threads hits another higher threshold, the computing node will enter a "dead" state. There is a "dead" threshold on the number of applications in each type on a computing node. The value of the "dead" threshold depends on the number of CPU cores, memory and I/O. For example, the "dead" threshold of applications in type C on an M4336 node is 9.

V. CONCLUSION

In cloud computing environments, the requirements on big data processing and application services have become more complex and diverse. In this paper, we proposed to create a multi-granular application layer (AaaS) and develop an application management platform (AaaS platform) between PaaS and IaaS in the paper. Moreover, we define an application model and present a parallel scheduling model and its policies. From the definitions, fine-grained application units (Apps) can be combined to create multi-granular applications in the AaaS platform. These applications can be scheduled by a multi-core-aware parallel scheduling model, which not only increases the system flexibility and applicability, but also improves the resource utilization of the fundamental infrastructure under fine-grained resource allocation. From the experimental results, we observe that the scheduling model can appropriately assign applications to computing nodes in a balanced manner, fully utilize the CPU resources on computing nodes and coordinate the collaboration between computing nodes, CPUs on a computing nodes and CPU cores for higher system performance and throughput.

ACKNOWLEDGMENT

This projectis supported by the Funds of Core Technology and Emerging Industry Strategic Project of Guangdong Province (Project No.: 2011A010801008, 2012A010701011, 2012A010701003); Guangdong Provincial Treasury Project (Project No.: 503-503054010110), Technology and Emerging Industry Strategic Project of Guangzhou(Project No.: 201200000034).

REFERENCES

[1] Armbrust M, Fox A, Griffith R, et al. A view of cloud computing[J]. Communications of the ACM, 2010, 53(4): 50-58.

[2] Cusumano M. Cloud computing and SaaS as new computing platforms[J]. Communications of the ACM, 2010, 53(4): 27-29.

[3] Kang S, Kang S, Hur S. A design of the conceptual architecture for a multitenant saas application platform[C].Computers, Networks, Systems and Industrial Engineering (CNSI), 2011 First ACIS/JNU International Conference on. IEEE, 2011: 462-467.

[4] Weissman C D, Bobrowski S. The design of the force.com multitenant internet application development platform[C].SIGMOD Conference. 2009: 889-896.

[5] Jin H, Ibrahim S, Bell T, et al. Cloud types and services[M]. Handbook of Cloud Computing. Springer US, 2010: 335-355.

[6] Azeez A, Perera S, Gamage D, et al. Multi-tenant SOA middleware for cloud computing[C]. Cloud computing (cloud), 2010 ieee 3rd international conference on. IEEE, 2010: 458-465.

[7] Pathirage M, Perera S, Kumara I, et al. A multi-tenant architecture for business process executions[C]. Web services (icws), 2011 ieee international conference on. IEEE, 2011: 121-128.

[8] Jin H, Ibrahim S, Bell T, et al. Cloud types and services[M]. Handbook of Cloud Computing. Springer US, 2010: 335-355.

[9] Lawton G. Developing software online with platform-as-a-service technology[J]. Computer, 2008, 41(6): 13-15.