Dynamic Advance Reservation for Grid System Using Resource Pools

Zhiang Wu, Jie Cao, and Youquan Wang

Jiangsu Provincial Key Laboratory of E-Business, Nanjing University of Finance and Economics, Nanjing, P.R. China
{zawuster,youq.wang}@gmail.com, caojie690929@163.com

Abstract. Dynamic behavior of resources is a non-negligible feature in grid system, and most research efforts on advance reservation cannot effectively deal with the negative effect resulted from the dynamic feature. In this paper, a new grid system architecture using resource pool is proposed firstly. Theoretical analysis demonstrates that resource pool can well adapt to dynamic behavior of resources. Secondly, Quality of Service (QoS) distance computation method for hybrid variable types is presented. Then, k-set Availability Prediction Admission Control (kAPAC) algorithm is described in detail. Experimental results show that kAPAC can significantly increase success ratio of reservation, resource utilization and stability of grid system.

Keywords: grid computing, advance reservation, resource pool, QoS.

1 Introduction

The last decade has witnessed tremendous progress of various distributed computing infrastructures aiming to support Internet-wide resources collaboration and sharing. Grid technology flourished worldwide in the early 2000's, and grid community requires participants to exhibit some degree of trust, accountability, and opportunities for sanctions in response to inappropriate behavior. This relative close community facilitates Quality of Service (QoS) guarantee mechanism.

Virtualization, a software-based technology for building shared hardware infrastructures in Grid computing, helps to achieve greater system utilization while lowering total cost of ownership and responding more effectively to changing business conditions [1]. Cloud computing also employs virtualization technology to provide dynamic resource pool. Therefore, virtualization will be a widely used technology for next generation computing platforms. Various resources are virtualized as a resource pool, and are managed as a whole.

In the meanwhile, with the advent of Web services, Internet computing (i.e. grid, cloud) has become a use case for Web Services. World is modeled as a collection of services: computational resources, storage resources, networks, programs, databases, and the like are all represented as services [2]. Seamless QoS (Quality of Service) is required to be delivered by this convergence between the architecture of grid system and service-oriented architecture. Google and IBM

E. Altman and W. Shi (Eds.): NPC 2011, LNCS 6985, pp. 123–134, 2011.

have proposed WS-resource framework (WSRF) to support stateful web services widely applied in Internet computing [3].

Advance reservation is a well-known and effective mechanism to guarantee QoS. Grid Resource Agreement and Allocation Protocol (GRAAP) work group of Global Grid Forum(GGF) has defined advance reservation [4]: an advance reservation is a possibly limited or restricted delegation of a particular resource capability over a defined time interval, obtained by the requester from the resource owner through a negotiation process. Advance reservation in grid environment is a complex issue due to the following reasons: (i) Since availability and performance of resources both exhibit dynamic variability, it is difficult to guarantee availability of resources at reserved time in the future. (ii) To satisfy users' requirements strictly often leads to an increase of miss-reject ratio, because some secondary requirements determine whether this reservation request will be admitted or not.

This paper focuses on dynamic advance reservation for the grid system utilizing resource pools. Firstly, three kinds of Service Level Agreements (SLAs) are introduced for QoS negotiation. Secondly, enabling system architecture employing resource pools to alleviate negative influence of grid dynamic behaviors is presented. Then, a new admission control approach, called k-Set Availability Prediction Admission Control (kAPAC), is proposed. At last, we demonstrate the effectiveness and efficiency of our kAPAC algorithm in experiments.

2 Related Work

Foster et al. propose general-purpose architecture for reservation and allocation (GARA) in early stage [5]. GARA supports reservation and adaptation, which simplifies the development of end-to-end QoS management strategies in service-oriented grid environment. This is the initial work on grid architecture to support QoS. Li et al. propose layered QoS scheduling aiming to maximize global user satisfaction degree at application layer [6]. Siddiqui et al. introduce 3-layered negotiation protocol for the advance reservation also aiming to achieve global optimization of application utility [7]. Service Negotiation and Acquisition Protocol (SNAP) is proposed in [8], which provides lifetime management and an at-most-once creation semantics for remote SLAs (Service Level Agreements). Three different types of SLAs are included in SNAP. They are task service level agreements (TSLAs), resource service level agreements (RSLAs) and binding service level agreements (BSLAs). We define a new SLA and extend states transition among SLAs in this paper.

Grid resources exhibit dynamic availability due to the unexpected failure, or due to dynamic joining and leaving and also exhibit dynamically varying performance due to the varying local load and unpredictable latency of today's Internet [9]. Our work takes this feature into consideration and utilizes resource pool to alleviate the negative influence of dynamic behavior. Resource pool is proposed in [10] and its performance is analyzed using queuing theory. This performance analysis method is expanded in this paper.

3 SLAs for QoS Negotiation

In our architecture proposed in Section 4, reservation requests are not bound with resources directly, but with resource pool. Three SLAs defined by SNAP cannot meet this requirement. We should define new SLA to support the dynamic binding between reservation request and resource pool.

A new SLA called Virtual Binding Service Level Agreement (VBSLA) is introduced, and states transition among SLAs is extended as shown in Fig. 1. Three kinds of SLAs defined by SNAP are still used in this paper. TSLA is used to negotiate for the performance of a task, which is characterized in terms of its service steps and QoS requirements. RSLA is used to negotiate for the right to consume a resource and each resource is characterized in terms of its service capabilities. BSLA associates a TSLA with the RSLA and the resource service capabilities should satisfy the task's requirements. In Fig. 1, S0, S1, S3 and S4 are original states in SNAP, and S2 is extended state for VBSLA. User submits reservation request at S1, and establishes VBSLA with resource pool at S2. When the start time of advance reservation request reaches, BSLA is established with certain resource at S3, and SLA state migrates to run-state S4.

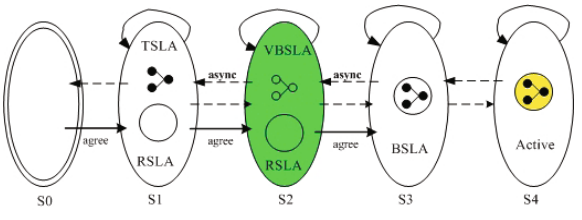

Fig. 1. Extended SLA States Transition

4 Grid Architecture Using Resource Pool for Advance Reservation

In this section, we first present a grid system architecture using resource pool. Then, we conduct theoretical analysis for resource pool.

4.1 Grid System Architecture

Resource pool aggregates resources provided by multiple physical sites. Physical sites control the amount of resources contributed to resources pool. The architecture of a resource pool supporting advance reservation is illustrated in Fig. 2. Admission control manager makes decisions about whether to accept a reservation request based on kAPAC algorithm which will be discussed in Section 5. Central pool sensor negotiates with local resource managers (LRM) of sites to determine the kind and ratio of resources contributed to the resource pool. Central pool actuator monitors the status of resource pool and gathers data for grid information center.

When reservation request is directly bound with specific resource, the reject rate of reservation requests will increase in a great extent, because the availability and performance of this bound resource cannot be guaranteed at runtime. Now, reservation request is firstly bound with resource pool at its arrival time, and this request will be bound with resource at its runtime according to the latest resource information. This dynamic binding method will alleviate negative influence of grid resources' dynamic behavior.

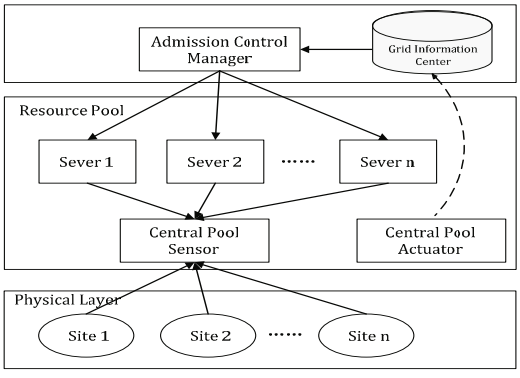

Fig. 2. Advance Reservation Architecture of a Resource Pool

4.2 Theoretical Analysis

It is obvious that little resources in resource pool cannot effectively guarantee QoS, but a large number of resources in resource pool may lead to resource profligacy. Let the arrival process of reservation requests be a *Poisson process* with an arrival rate λ. Assume service time conforms any probability distribution, which is the general form. Then, a resource pool aggregating c resources is modeled as $M/G/c$ queuing system. The average waiting time of $M/G/c$ is given by Eq.(1).

$$W_c = \frac{\overline{h}^2}{2\overline{h}(c-\lambda\overline{h})}[1+\sum_{n=0}^{c-1}\frac{(\lambda\overline{h})^n}{n!}\frac{(c-1)!(c-\lambda\overline{h})}{(\lambda\overline{h})^c}]^{-1} \tag{1}$$

In Eq.(1) $\overline{h}$ is the first moment of service time and $\overline{h}^2$ is the second moment of service time. Since file length in Web server and response time of websites are deemed to conform a heavy-tailed distribution [11], we assume that service time conforms *Bounded Pareto* (BP) distribution which is a typical heavy-tailed distribution. The probability density function of BP distribution is shown in Eq.(2).

$$f(x) = \frac{\alpha k^\alpha}{1-(k/p)^\alpha}x^{-\alpha-1} \qquad k \leqslant x \leqslant p \tag{2}$$

The first moment and second moment of BP distribution can be computed by Eqs. (3) and (4).

$$\overline{h} = \frac{\alpha}{\alpha - 1} \frac{k^{\alpha}}{1 - (k/p)^{\alpha}} \left(\frac{1}{k^{\alpha-1}} - \frac{1}{p^{\alpha-1}} \right) \tag{3}$$

$$\overline{h}^2 = \frac{\alpha}{\alpha - 2} \frac{k^{\alpha}}{1 - (k/p)^{\alpha}} \left(\frac{1}{k^{\alpha-2}} - \frac{1}{p^{\alpha-2}} \right) \tag{4}$$

Table 1 lists five cases with different k and p values. We set α to 1.1 and $p = 10000 * k$. We reveal the relation between the average waiting time and the number of resources in five cases as shown in Fig. 3.

Table 1. Experiment Parameters (k,p) and The Mean and Variance of BP Distribution

	Case 1	**Case 2**	**Case 3**	**Case 4**	**Case 5**
k	0.1	1	10	30	50
p	1000	10000	100000	300000	500000
Mean	0.66211	6.6211	66.211	198.63	331.05
Variance	48.209	4820.9	20900000	388000000	520000000

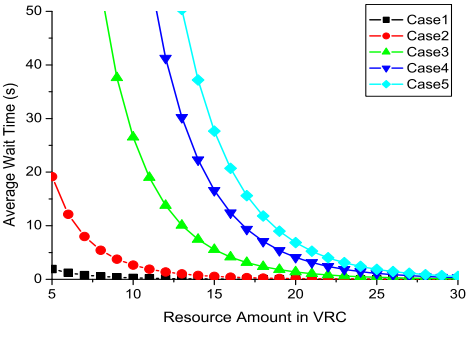

Fig. 3. Resource Amount in VRC vs. Average Wait Time

Fig.3 shows that the average waiting time decreases dramatically at the beginning, but once resource amount reaches threshold, average waiting time decreasing rate reduces in evidence. This threshold is inflection point of this group of curves, and from theoretical view, it is optimal resource amount provisioned in resource pool. Since Eq. (1) is non-continuous, we can not obtain precise X-coordinate of its inflection point. We just design an algorithm to find approximate optimal amount of resources. Approximate optimal amount is the first point that decreasing extent between two neighbor points reaches threshold. This threshold is set as average wait time that most users can endure. We observe kept W_c unchanged (means non-degraded QoS level), the permitted workload ratio (means service ability) will be increased with the increase of amount of resources integrated by resource pool. Therefore, the resource pool can avoid negative influence of grid resource dynamic fluctuation effectively, but without degrading the performance of advance reservation.

5 *k*APAC Algorithm

Admission control manager aims to determine whether grid resource pool accepts reservation requests. Most systems compare RSLA with TSLA, and only when all QoS requirements specified in TSLA are satisfied by RSLA, request can be accepted by system. The accepted requests are bound with fixed resources. There are two significant limitations in this method. First, some secondary QoS attributes determine requests whether to be accepted or not. The success rate of reservation requests decreases in extreme extent. Second, *availability* of resources are not considered. In fact, resources will be utilized by reservation requests in future time. Therefore, resources with low availability leads to the failure of the bound requests in a high probability.

This section presents a new admission control algorithm named kAPAC. QoS distance computation method is presented firstly. Then, the procedure of kAPAC is proposed.

5.1 QoS Distance Computation

Variable types of QoS attributes are widely different. For example, response time, bandwidth and delay are interval-scale variables; the level of security is discrete ordinal variable enumerating *low, middle, high* in-order. QoS distance computation method proposed in this paper deals with these hybrid variable types in a uniform framework. Let Q_i and Q_j denote two QoS vectors, and $d(Q_i, Q_j)$ is given by Eq. (5).

$$d(Q_i, Q_j) = \frac{\sum_{f=1}^{p} \delta_{ij}^f d_{ij}^f}{\sum_{f=1}^{p} \delta_{ij}^f} \tag{5}$$

In Eq.(5), if Q_i or Q_j do not assign a value to the f-th variable (x_{if} or x_{jf} are missing), δ_{ij}^f=0; otherwise δ_{ij}^f=1. The distance of the f-th variable between Q_i and Q_j is written d_{ij}^f which relies on the variable type. We consider binary variable, categorical variable, interval-scale variable and discrete ordinal variable respectively.

- the f-th variable is binary variable or categorical variable: if $x_{if} = x_{jf}$, d_{ij}^f =0; otherwise d_{ij}^f =1.
- the f-th variable is interval-scale variable: can be computed by Eq. (6).

$$d_{ij}^f = \frac{|x_{if} - x_{jf}|}{max_h x_{hf} - min_h x_{hf}} \tag{6}$$

- the f-th variable is discrete ordinal variable: assume the f-th variable has M_f states. We define ranking 1,2,$\cdots$,M_f for these M_f states, and the ranking corresponding to x_{if} is written r_{if}. Then, we convert r_{if} to interval-scale variable which is written z_{if}. z_{if} is given by Eq.(7), and Eq.(6) can be utilized to compute d_{ij}^f.

$$z_{if} = \frac{r_{if} - 1}{M_f - 1} \tag{7}$$

Moreover, we suppose there exists m service QoS parameters and namely p-m provisional QoS parameters. Let h_i denote whether all service QoS parameters of task request are satisfied perfectly. Adjusted distance between Q_i and Q_j is written $d'(Q_i, Q_j)$.If h_i =0, this reservation request cannot be satisfied and $d'(Q_i, Q_j)$ is infinite. $d'(Q_i, Q_j)$ is computed by Eq. (8).

$$d'(Q_i, Q_j) = \begin{cases} +\infty, h_i = 0 \\ d(Q_i, Q_j), h_i = 1 \end{cases} \tag{8}$$

With Eqs. (5) to (8), we can compute distance between reservation request and resource and distance between two requests.

5.2 Algorithm Design

kAPAC algorithm aims to accept reservation requests which will be executed in the future in a high probability. The data structure $RAQueue$ holds the requests that have been accepted but do not be executed. The pseudo-code of kAPAC algorithm is presented in Table 2.

Table 2. Pseudo-code Describing the kAPAC algorithm

kAPAC is deployed to Admission Control Manager, and is used to determine whether to accept a reservation request or not
1: **while** ($TRUE$) **do**
2: wait_request(q) //reservation request q arrives
3: compute the number of similar requests with q in $RAQueue$, which is written $Index_{rej}$
4: **if** ($Index_{rej} >= k$) **then** //the similar requests are enough to reject q
5: reject_request(q)
6: **else**
7: select k nearest neighbors (kNN) of q in resource set
8: compute the availability of q's kNN, which is written $availability\_k\_set(q)$
9: **if** ($availability\_k\_set(q) > threshold_{acp}$) **then**
10: accept_request(q) and $RAQueue$.enqueue(q)
11: **end of while**

The process of kAPAC is that: when a reservation request arrives, it first selects top-k resources with small distance as much as possible to form k-set; then it judges whether availability of k-set resources is bigger than accept threshold; if it is, system accepts this task. But when lots of similar task requests arrive, selected k-set will also be similar. If availability of this k-set is bigger than accept threshold, these similar task requests are all accepted. Since these accepted tasks will occupy lots of similar resources in future and last for an interval, when similar task requests increase much more than k, these k resources will be busy in future in great probability and surplus requests will be aborted at runtime.

To avoid this case, kAPAC firstly judges whether the number of similar requests in $RAQueue$ is bigger than k. If it is, system aborts this reservation request.

To facilitate our study, we utilize a simplified model to predict resource availability. This model conforms to two observations: (i) Availability of recent used resource will be much higher in the future, and thus future availability is estimated by both historical and recent information. (ii) Historical availability decreases with the increase of idle time. We use recent availability A_{recent} to represent recent information and use $A_{cur} * g(t)$ to represent historical information. $g(t)$ is a non-incremental function here and t is the interval from last used time. Recent availability and current availability are calculated by Eqs. (9) and (10) respectively.

$$A_{recent} = \frac{T_{active}}{T_{total}} = \frac{T_{active}}{T_{active} + T_{down}} \tag{9}$$

$$A_{cur} = \omega \cdot A_{cur} \cdot g(t) + (1 - \omega) * A_{recent} \quad 0 < g(t) < 1 \tag{10}$$

Every time when resource is used, Eq. (9) is used to measure most recent availability, in which T_{active} is total time when this resource is active (not down) and of course T_{total} equals sum of T_{active} and T_{down}. Eq. (10) is used to estimate current availability. Current availability is weighted sum of historical availability and recent availability, the right A_{cur} in Eq. (10) is the old current availability that is calculated last used time. Current availability decreases with the increase of t.

6 Experimental Results

Our experiment is conducted in Southeast University Grid (SEUGrid) developed based on Globus Toolkit [12]. SEUGrid is designed for Alpha Magnetic Spectrometer (AMS-02) experiment. The Monte Carlo (MC) production, an important jobs in AMS experiment, is a kind of typical parameter sweep application. There are also no inter-task communication or data dependencies in MC production. We utilize part of computational resources in SEUGrid to conduct our experiments, containing two clusters each with 32 blade servers and one PowerLeader server. The total number of CPU reaches 140.

To simulate the dynamic performance fluctuation, all grid nodes produce local *High Performance Linpack* (HPL) test jobs randomly to form local workload. Thus, system workload ratio can be easily estimated according to size of HPL jobs. Another type of job, MC production jobs including various QoS constraints, submits advance reservation request to Admission Control Manager. All reservation requests mentioned below refer to MC production jobs. We firstly define some evaluation metrics that will be used in our experiments.

1. W_r **(Workload Ratio):** This metric is used to measure workload of overall system in a given time interval. It is determined by mount and size of HPL jobs within an interval, and can be calculated by Eq. (11).

$$W_r = \frac{\sum_{i=1}^{m}(\frac{2}{3} \cdot N_i^3 - 2 \cdot N_i^2)}{t \cdot \sum_{i=1}^{n} Rpeak_i} \tag{11}$$

Where N_i is the size of one HPL job, and $2/3 * N_i^3 - 2 * N_i^2$ is its actual calculation flops, which is defined in HPL specification. $Rpeak_i$ is theoretical peak of one CPU, and the denominator is total peak of system during a time interval.

2. R_s **(Success Ratio):** This metric is the percentage ratio of reservation requests that are successfully completed.
3. R_a **(Accept Ratio):** This metric is the percentage ratio of reservation requests that are accepted by admission control manager.
4. E_r **(Effectiveness Ratio):** This metric is the average percentage ratio of successfully completed requests in total accepted requests. R_s is always smaller than R_a since part of accepted requests may be aborted at their start time due to sudden unavailability of estimated available resources, or due to resource contention. E_r reflects how many accepted requests are aborted at start time. The less accepted requests are aborted, the higher E_r is. High E_r shows that this system performs good effectiveness and brings good trust to users. In contrast, bad system effectiveness may lost their users.
5. U_r **(Utilization Ratio):** This metric is the average percentage ratio of total resources that used by tasks, and can be calculated by Eq. (12).

$$U_r = \frac{\sum_{i=1}^{m} Request(i)}{(1 - W_r) \cdot t \cdot \sum_{i=1}^{n} Rpeak_i} \tag{12}$$

Where $Request(i)$ is the computation amount of a MC production job. W_r is the workload ratio of overall system in a time interval, and so the denominator is actual computation amount provided by system during this time interval.

6.1 Impact of *k*-set Length and Recent Queue Length

In kAPAC, recent queue length L is set as a times of k-set length k. That is $L = a * k$. In this experiment, we investigate how a and k affect on U_r and E_r. We range k from 1 to 10 and set a to 3, 2, and 1.5 respectively. We also set average request inter-arrival time to 80s. Fig. 4 and Fig. 5 show U_r and E_r obtained by increase of k from 1 to 10 with different a, in which each plots is measured in 30 minutes.

Comparing the plots both in Fig. 4 and Fig. 5 indicates that with the increase of k, U_r increases but E_r decreases. It also indicates that the increase of a brings the same results as the increase of k. Since with the increase of k, grid system becomes more open and more requests can be admitted, causing that resources become more busier and resource contention often happens. Thus, U_r will increase. But the opening characteristic also results that more admitted requests will be aborted at their start time due to resource contention, and then system effectiveness ratio E_r will reduce. Therefore, we conclude that open

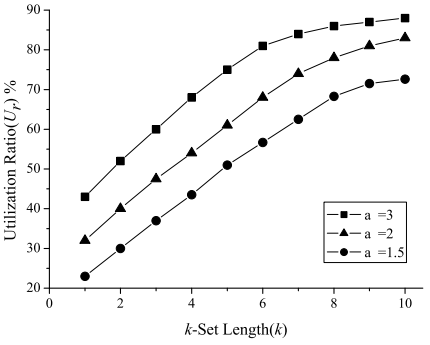

Fig. 4. k-set length (k) vs. U_r

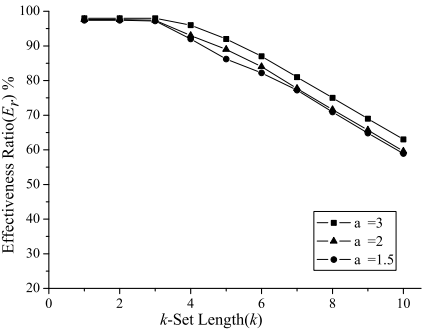

Fig. 5. k-set length (k) vs. E_r

system has high accept ratio and high resource utilization, but has low system effectiveness ratio. Conversely, conservative system which sets k and a to small values has high system effectiveness ratio, but low accept ratio and low resource utilization. Thus, k and a have to be carefully chosen to achieve good tradeoff between U_r and E_r.

6.2 *k*APAC vs. SNAP Strict Negotiation

Most advance reservation architectures compare TLSA with RSLA strictly, called SNAP strict negotiation in this paper. If QoS requirements of a request specified in TSLA can be satisfied by one resource, this request can be admitted. The difference with kAPAC is that this method only considers one resource to satisfy reservation request and compares QoS requirements strictly but not based on QoS Euclidian distance. We assume reservation request arrival process of MC production is a Poisson process, and vary average request inter-arrival time from 500 to 20s. And we set k to 5 and a to 2. Fig. 6 and Fig. 7 compare success ratio S_r and utilization ratio U_r between kAPAC and strict negotiation respectively, whose plots are also measured in 30 minutes. When request arrival rate becomes high, Fig. 6 shows that although S_r reduces in both two cases, kAPAC has higher S_r than strict negotiation all the time and also reduces less slowly. Since when resource contention becomes not negligible, kAPAC can choose resources in great range for it takes QoS Euclidian distance as metric, which satisfies service QoS parameters strictly and bears degradation of provisional QoS parameters. Fig. 7 indicates that kAPAC also leads higher U_r than strict negotiation and brows up quickly when request arrival rate increases. In strict negotiation once one resource can satisfy reservation request this request is admitted, but lack of consideration on availability of this resource. Thus, more requests will be aborted after they are admitted, which results the decrease of U_r.

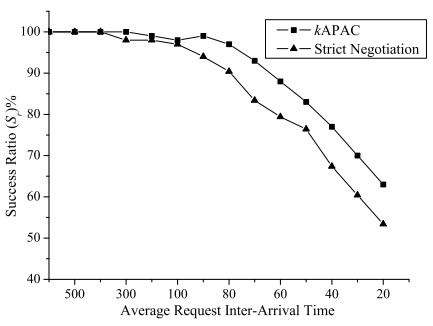

Fig. 6. Comparison between k-APACA and Strict Negotiation on S_r

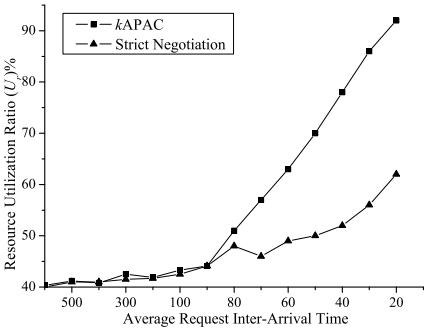

Fig. 7. Comparison between k-APACA and Strict Negotiation on Resource U_r

7 Conclusion

In this paper, grid system architecture using resource pool is proposed and theoretical analysis for resource pool is conducted. Theoretical analysis demonstrates that the number of resources in pool should be moderate and an algorithm determining approximate optimal resource amount is proposed. Since variable types of QoS attributes are widely different, a uniform framework for computing QoS distance is presented. QoS distance can be utilized to compare the similarity of two requests or between request and resource. kAPAC algorithm proposed in this paper proceeds based on QoS distance comparison. Experiments are conducted in SEUGrid designed for AMS-02 experiment. Comparison between kAPAC and SNAP strict negotiation shows that kAPAC can increase success ratio of reservation, resource utilization and stability of grid system remarkably.

Acknowledgments. This research is supported by National Natural Science Foundation of China under Grants No.71072172, National Soft Science Foundation of China under Grants No. 2010GXS5D208, Jiangsu Provincial Key Laboratory of Network and Information Security (Southeast University) under Grants No.BM2003201, Transformation Fund for Agricultural Science and Technology Achievements under Grants No. 2011GB2C100024 and Innovation Fund for Agricultural Science and Technology in Jiangsu under Grants No. CX(11)3039.

References

1. Gmach, D., Rolia, J., Cherkasova, L., Kemper, A.: Resource Pool Management: Reactive Versus Proactive or Let's be Friends. Computer Networks 53, 2905–2922 (2009)
2. Malik, Z., Rater, B.A.: Credibility Assessment in Web Services Interactions. World Wide Web: Internet and Web Information Systems 12(1), 3–25 (2009)

3. Czajkowski, K., Ferguson, D.F., Foster, I., et al.: The WS-resource Framework, Version 1.0 (2004), http://www-106.ibm.com/developerworks/library/ws-resource/ws-wsrf.pdf
4. MacLaren, J.: Advance Reservations: State of the Art, GGF GRAAP-WG (2003), http://www.fz-juelich.de/zam/RD/coop/ggf/graap/graap-wg.html
5. Foster, I., Roy, A., Sander, V.: A Quality of Service Architecture That Combines Resource Reservation and Application Adaptation. In: International Workshop on Quality of Service, pp. 181–188 (2000)
6. Li, C., Li, L.: Utility -based Scheduling for Grid Computing under Constraints of Energy Budget and Deadline. Computer Standards & Interfaces 31(6), 1131–1142 (2009)
7. Siddiqui, M., Villazon, A., Fahringer, T.: Grid Capacity Planning with Negotiation-based Advance Reservation for Optimized QoS. In: Proceedings of the ACM/IEEE Supercomputing Conference, Tampa, Florida, USA, pp. 21–36 (2006)
8. Czajkowski, K., Foster, I., Kesselman, C., Sander, V., Tuecke, S.: SNAP: A Protocol for Negotiating Service Level Agreements and Coordinating Resource Management in Distributed Systems. In: Feitelson, D.G., Rudolph, L., Schwiegelshohn, U. (eds.) JSSPP 2002. LNCS, vol. 2537, pp. 153–183. Springer, Heidelberg (2002)
9. Liu, Y.H., Liu, X.M., Ni, L.M., Zhang, X.D.: Location-Aware Topology Matching in P2P Systems. In: Proceedings of IEEE INFOCOM 2004, pp. 2220–2230 (2004)
10. Wu, Y., Yang, G., Mao, J., Shi, S., Zheng, W.: Grid Computing Pool and Its Framework. In: International Conference on Parallel Processing Workshops (2003)
11. Subrata, R., Zomaya, A.Y., Landfeldt, B.: Game-Theoretic Approach for Load Balancing in Computational Grids. IEEE Trans. on Parallel and Distributed Systems 19(1), 66–76 (2008)
12. Luo, J., Song, A.-B., Zhu, Y., Wang, X., Ma, T., Wu, Z.-A., Xu, Y., Ge, L.: Grid Supporting Platform for AMS Data Processing. In: Chen, G., Pan, Y., Guo, M., Lu, J. (eds.) ISPA-WS 2005. LNCS, vol. 3759, pp. 276–285. Springer, Heidelberg (2005)