Resource Calculations with Constraints, and Placement of Tenants and Instances for Multi-tenant SaaS Applications

Thomas Kwok and Ajay Mohindra

IBM Research Division
Thomas J. Watson Research Center
19 Skyline Drive, Hawthorne, NY 10532
{kwok,ajaym}@us.ibm.com

Abstract. Cost of customization, deployment and operation of a software application supporting multiple tenants can be lowered through multi-tenancy in a new application business model called Software as a Service (SaaS). However, there are a number of technical challenges that need to be tackled before these benefits can be realized. These challenges include calculations of resource requirements for multi-tenants with applied constraints in a shared application instance, the optimal placement of tenants and instances with maximum cost savings but without violating any requirements of service level agreements for all tenants in a set of servers. Moreover, previously reported capacity planning and resource allocation methods and tools are not tenant aware. This paper will address and provide novel solutions to these challenges. We also describe the first of a kind, a multi-tenant placement tool for application deployment in a distributed computing environment.

Keywords: capacity planning, resource allocation and management, tenant placement, constraint, multi-tenant, software as a service, SaaS.

1 Introduction

Recently, a new business model of software applications called Software as a Service (SaaS), which offers benefits of lower cost of customization, deployment and operation over the Internet has evolved [1-3]. In general, SaaS is associated with business software applications that are Web-based, deployed and operated as a hosted service accessed by users over the Internet. Multi-tenants in addition to multi-users support, installation of application on a managed Internet data center with remote management capability are a few characteristics of a multi-tenant SaaS application [4-8]. In the SaaS business model, the ownership of technology infrastructure and management responsibility of the application has moved to application service providers (ASPs) from tenants. The multi-tenant SaaS model benefits ASPs by reducing hosting costs as the same application is shared among multi-tenants. It also benefits tenants through eliminating their costs of owning and managing the infrastructure and applications. Tenants can gain immediate access to the latest information technology (IT) innovations and improvements provided by the

A. Bouguettaya, I. Krueger, and T. Margaria (Eds.): ICSOC 2008, LNCS 5364, pp. 633–648, 2008.

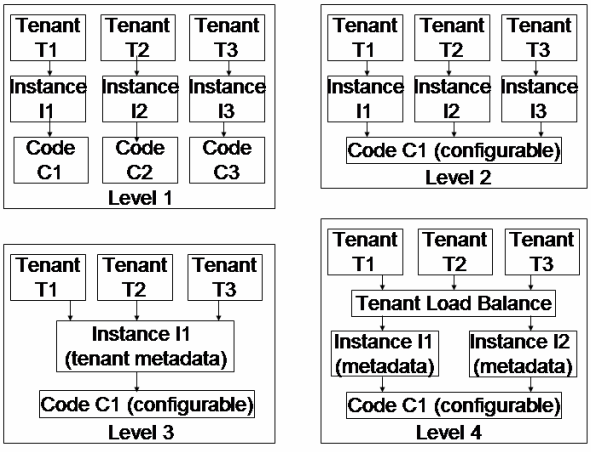

Fig. 1. Four levels of the multi-tenant support model in an application layer

ASP without spending their own IT budgets. In the SaaS business model, usages of the application can either be on a per user basis or pay as you go basis.

In a multi-tenant SaaS model, the multi-tenant support can be applied to four different software layers: application, middleware, virtual machine (VM) and operating system [4]. For the application layer, there are four levels of the multi-tenant support model as shown in Figure 1 [5]. Level 1 has a separate instance for each tenant's customized code base and it is similar to the ASP model. Level 2 also has a separate instance for each tenant but all instances come from a single code base with configuration options. With a single application code base to support several tenants, the total deployment time is shorter. Level 3 has a single instance for all tenants with configurable metadata for each tenant. In this level, the updating of application features and functions are simpler and centralized because there is only one instance of a single source code base. Level 4 has a load-balanced farm of identical instances with configurable metadata for its tenants. There are many benefits of multi-tenant application deployment in Levels 3 and 4. The main benefits are the reduction of IT expenses, and cost savings in software license fees and hardware expenses by reducing the number of development, staging, training, disaster recovery servers. Other benefits are in deployment, provision, on-board and maintenance by reducing IT processes, such as server and application setup and configuration, and reducing support staffs in server and application tuning, backup, patch and upgrade. Costs in cooling and HVAC, power and lighting are reduced due to fewer servers [9].

However, several initial setup and configuration steps have to be carried out in order for the application to support multi-tenants in a SaaS operational structure [8]. There are also a number of challenges that require solutions before full benefits of multi-tenant application deployment can be realized. First, it is difficult to calculate resource requirements for each additional new tenant with a number of new users, and at the same time meeting constraints for all tenants in a shared application instance. Second, limiting factors or bottlenecks on computing resources required for multiple instances, each with multi-tenants having different constraints, have to be determined.

Third, an administrator needs the advice on the placement of a group of multi-tenant applications on a set of servers without violating any service level agreement (SLA) requirements of all tenants. Fourth, the placement of tenants and instances in a distributed computing environment has to be automated. Fifth, cost savings among different multi-tenant placements have to be compared and optimized even though there are many variables involved. Other challenges are multi-tenant data models, tenant isolation and security related issues. This paper will address and provide novel solutions to the first four challenges. We also describe the first of a kind, a placement tool for multi-tenant application deployment in the third level of the multi-tenant support model.

2 Prior Related Work

Capacity planning and resource allocations to satisfy application requirements, and resource constrained project scheduling are a generalization of the static job shop problem, and have been reviewed thoroughly by researchers [10-12]. Most of these studies are based on traditional exact methods, priority rule schedule [13] and meta-heuristic approach [14]. A priority rule schedule consists of two parts, a priority rule and a scheduling scheme. In the meta-heuristic approach, an activity list is usually first created. Then, a neighboring schedule is identified by changing the order of tasks in the list. A quality of service (QoS) based resource allocation model has also been used to make sure that different constraints of concurrently running applications are satisfied [15]. However, all these research reports are not tenant aware, and do not take into account characteristics of multi-tenant application with a single instance supporting multi-tenants. Calculations of computing resources, such as central processing units (CPU) and memory, required for an instance supporting multi-users are rather straight forward and simple [10]. Calculations of resources required for an instance supporting multi-tenants with multi-users in each tenant are new and complicated. Up to now, there is no reported multi-tenant resource data model that can be used to calculate resource requirements for multi-tenants in a shared instance. Again, calculations of the maximum number of users in an instance on any server of specific resources without violating any SLA requirements for a single tenant has been outlined [11]. Calculations of maximum numbers of users and tenants on a shared instance in any server of specific resources satisfying all constraints listed on SLAs of all tenants are new and complicated.

A hierarchical and extensible resource management system has been built to allow the execution of multiple scheduling paradigms concurrently [16]. A number of commercial software tools for capacity planning, resource allocation and performance analysis for multiple applications on a set of servers are also available [16,17]. However, these commercial tools do not apply to the placement of tenants and instances for multi-tenant SaaS applications. They primarily focus on the placement of applications to available servers based on physical resources. Since they do not know how much resource requirements for a shared instance with additional tenants and constraints, a new application instance is always created and deployed for each new tenant. Furthermore, most manual resource capacity estimates on servers that work satisfactorily are those that are oversized and thus more expensive [9].

3 Resource Calculations for Multi-users and Multi-tenants

An application can demand a number of computing resources, such as CPU, memory, storage disk and network bandwidth, from physical hardware of the host server in a distributed computing environment. There are several different characteristics of these computing resources. A hard disk is considered the primary permanent storage device. An application instance requires an initial amount of storage, such as initialization of tables. These tables are shared among tenants and users. Additional amount of storage is required to store data for each additional tenant or user, such as adding additional tables and/or rows in different tables. Thus, storage usage can be assumed to be load dependent and proportional to numbers of tenants or users. Similarly, significant amount of memory in dynamic random access memory (DRAM) is consumed by an instance even if there is no active tenant or user. There are paged and non-paged memory. Non-paged memory consists of a range of virtual addresses guaranteed to be in DRAM at all times, and paged memory can be swapped to slower system resources, such as hard disk. As a result, it is very difficult to accurately project memory usage based on the number of tenants and users in a shared instance. Above all, many applications cannot run when the system is out of memory. Thus, only the maximum memory usage can be assumed slightly dependent on the number of tenants and users. Hence, an estimated upper limit on memory usage is often used. In some advance VMs, each instance may be able to use multiple CPUs if the host hardware physically contains multiple CPUs. Unlike storage and memory, CPU usage with same clock speed and same amount of Levels 1, 2 and 3 static RAM (SRAM) caches can be assumed to be linearly proportional to the number of active tenants and users because the processing speed of a CPU depends on both clock speed and cache.

For practical reasons, CPU and storage are used to illustrate calculations of resource requirements in this paper. For an instance supporting multi-users, calculations are rather straight forward and simple [10]. Let r be the number of users, and C(r) and M(r) be the CPU and storage required by an instance with multi-users, respectively. Then,

$$\begin{aligned} C(r) &= f_{CU}(r) . \\ M(r) &= M_0 + f_{MU}(r) . \end{aligned} \tag{1}$$

where $f_{CU}(r)$ and $f_{MU}(r)$ are functions of r, assuming that the CPU instance is idle if there is no active user. M_0 is a constant representing overhead storage used by the instance without any users. However, calculations of resources required for a shared instance supporting multi-tenants and multi-users are new and complicated. Let t be the number of tenants in a shared instance and n be the total number of users. Then,

$$\begin{aligned} C(n,t) &= f_{CU}(n) + f_{CT}(t) . \\ M(n,t) &= M_0 + f_{MU}(n) + f_{MT}(t) . \end{aligned} \tag{2}$$

where $f_{CT}(t)$ and $f_{MT}(t)$ are functions of t. These two functions are additional CPU and storage required to isolated tenants from each other in a shared instance. For a special case where there are two tenants $t = 2$ and the number of users in the two tenants are both equal to r such that $n = 2r$. Let us compare resources required in this

special case deployed in two different computing environments. First, in two application instances, each with a tenant and *r* users, and from Equations (1):

$$\begin{aligned} 2C(r,1) &= 2C(r) = 2f_{CU}(r) . \\ 2M(r,1) &= 2M(r) = 2M_0 + 2f_{MU}(r) . \end{aligned} \tag{3}$$

Second, in one application instance with two tenants and *r* users in each tenant, and from Equations (2):

$$\begin{aligned} C(2r,2) &= f_{CU}(2r) + f_{CT}(2) . \\ M(2r,2) &= M_0 + f_{MU}(2r) + f_{MT}(2) . \end{aligned} \tag{4}$$

Assuming $f_{CU}(r)$ and $f_{MU}(r)$ are linearly proportional to *r,* taking the first order approximation:

$$\begin{aligned} f_{CU}(2r) &= 2f_{CU}(r) . \\ f_{MU}(2r) &= 2f_{MU}(r) . \end{aligned} \tag{5}$$

Since a certain amount of storage is shared by both tenants, and additional amount of storage required to isolate tenants from each other is relative small in a shared instance, then:

$$f_{MT}(2) << M_0 . \tag{6}$$

According to Equations (5) and (6):

$$\begin{aligned} f_{CU}(2r) + f_{CT}(2) &> 2f_{CU}(r) . \\ M_0 + f_{MU}(2r) + f_{MT}(2) &<< 2M_0 + 2f_{MU}(r) . \end{aligned} \tag{7}$$

Thus from Equations (3), (4) and (7):

$$\begin{aligned} C(2r,1) &> 2C(r,1) . \\ M(2r,2) &<< 2M(r,1) . \end{aligned} \tag{8}$$

As a result, there are relatively large savings in those resources shared by multi-tenants, such as storage and memory, but at the same time a little bit more usage of resources, such as CPU and network bandwidth, for deploying multi-tenants in a shared instance. Accordingly, many tenants should be deployed in a shared instance instead of only one tenant per instance in a server for a multi-tenant SaaS application.

4 Resource Data Models for Multi-tenants in a Shared Instance

Up to now, there is no reported computing resource data model for multi-tenants in a shared application instance. Functions $f_{CU}(n)$, $f_{MU}(n)$, $f_{CT}(t)$ and $f_{MT}(t)$ can be in the form of curves or tables of measured data. Equations based partially on theory and partially on these empirical data can be obtained by fitting these curves or tables with interpolation or extrapolation algorithms [18]. These semi-empirical equations can be linear, polynomial, power, exponential, logarithmic or any other types depending on fractions of different activities, such as Web, computation, transaction and database, involved in the application. Hypothetical data of storage requirements as a function of

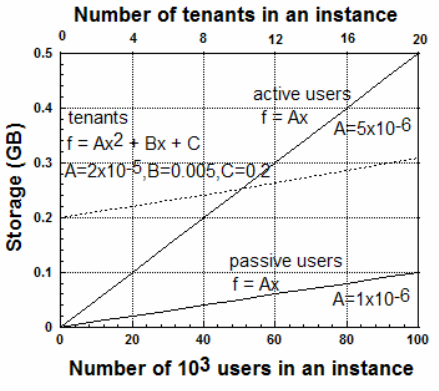

Fig. 2. Hypothetical data of storage requirements as a function of active and passive users, and tenants

users and tenants in a shared instance are shown in Figure 2. Assuming that storage usage by each user is independent from other users, and from the total number of users, semi-empirical parameters based on the first order approximation are obtained by fitting solid curves in Figure 2. However, storage usage by each tenant may increase with the total number of tenants in the shared instance because additional storage is required to isolate each tenant from the increasing number of other tenants. Thus, semi-empirical parameters based on the second order approximation are obtained by fitting the dotted curve in Figure 2.

As shown in Figure 2, passive users also demand storage usage but their usage is much less than that of active users. This is also true for memory usage. Let x be the concurrent user or peak load rate of an application instance and y be the utilization rate of a server. Lowering the utilization rate below 1.0 will provide higher service reliability and increase uptime, which will eliminate or reduce fines caused by missed SLA requirements. Let u and p be total numbers of active and passive users in a shared instance, respectively. Thus,

$$\begin{aligned} u &= n * x . \\ p &= n * (1.0 - x) . \end{aligned} \tag{9}$$

According to Equations (2) and (9), the total storage required by t tenants with total number of users n in a shared instance is given by

$$M(n,t) = (f_{MU}(u) + f_{MU}(p) + M_0 + f_{MT}(t)) / y . \tag{10}$$

where $f_{MU}(u)$ and $f_{MU}(p)$ are obtained from two solid curves while M_0 and $f_{MT}(t)$ are obtained from the dotted curve. As shown in Figure 2, M_0 is the intercept on the Y-axis when *t = 0 or 1* as an application instance either requires no tenant or a minimum of one tenant for initialization. Similarly, the total CPU required by t tenants with total number of users n in a shared instance is given by

$$C(n,t) = (f_{CU}(u) + f_{CT}(t)) / y . \tag{11}$$

assuming that the CPU instance is idle if there is no active user. Other computing resources, such as memory and network bandwidth, required by multi-tenants in a shared instance can be calculated in similar ways using either Equations (10) or (11).

5 Constraints on a Multi-tenant Application Instance

From previous two sections, calculated computing resources, such as CPU and storage, based on the number of users and tenants in a shared instance are the basic or minimum requirements of available resources in any server on which the shared instance would run. However, there are also a number of constraints on limiting the maximum number of users and tenants on this shared instance running in any server of specific resource. These constraints can be response time, availability, user arrival and page view rates, business transaction rate, request and transfer rates for database, input and output operational rates on file system, as well as disaster recovery, cost and reporting metrics, in SLA specifications. Operating the shared instance within these constraints will reduce or eliminate fines caused by missed SLA requirements.

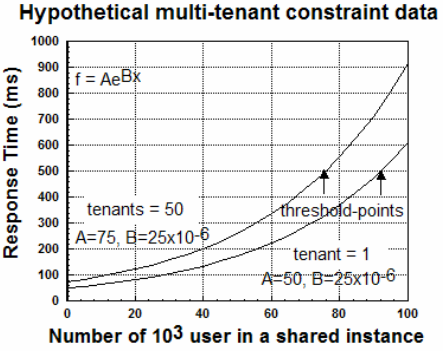

Fig. 3. Hypothetical data of response time as functions of user and tenant numbers

Again, calculations of the maximum number of users on an instance in any server of specific resources without violating any SLA requirements have been outlined [11]. However, calculations of maximum numbers of users and tenants on a shared instance running in any server of specific resources satisfying all constraints listed on SLA specifications of all tenants are new and complicated. For practical reasons, response time is used to illustrate calculations of resource requirements with applied constraints in this paper. Hypothetical data of response time limiting the maximum number of users and tenants on a shared instance in any server of specific resources is shown in Figure 3. Semi-empirical parameters based on the exponential approximation are obtained by fitting these two curves. Let the constraint on response time listed on SLA specifications be 500 ms, then the maximum number of users allowed in an instance with only 1 tenant is around 92×10^3 while that in a shared instance with 50 tenants is around 75×10^3. The maximum number of users on a shared instance with t tenants can be found by interpolation or extrapolation of these two curves [18]. The

maximum number of users and tenants allowed in a shared instance running on any server of specific resources for other constraints listed on SLA specifications of all tenants can be carried out in a similar way.

The algorithm for multi-tenant resource calculations with applied constraints is illustrated with block diagrams in Figure 4. There are three main parts of this algorithm. Let J be the total number of resource types and its index j is from *1 to J*. For practical reasons, only two resource types *(J=2)*, CPU C*(j=1)* and storage M*(j=2)*, are used to demonstrate multi-tenant resource calculations in this paper. Other resource parameters, such as network bandwidth and memory, can be added into these calculations in similar ways. Let S be the total number of available servers and its index s is from *1 to S*. In the first part, the multi-tenant active placement sensor will provide information on current resource usages of tenants T_i with users N_i in each i of shared instances I of applications A, as well as residual resources in available servers S. This first part is to calculate resource demands due to new tenants ΔT_i with new users ΔN_i on an active instance i of an application a in a specific server s. Let $C_{0i}(N+\Delta N,T+\Delta T)$ and $M_{0i}(N+\Delta N,T+\Delta T)$ represent two sets of resource demand vectors for minimum CPU and storage requirements due to additional tenants ΔT and additional users ΔN on an instance i of an application a in a server s. According to Equations (2),

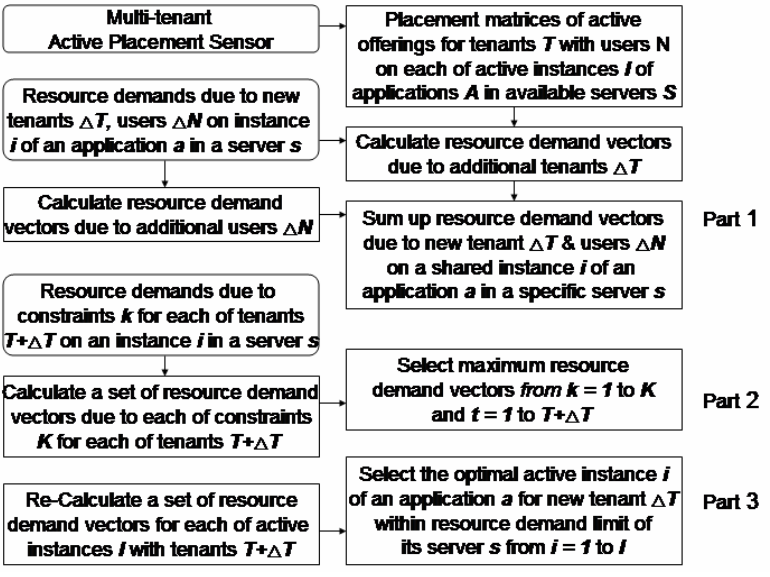

Fig. 4. An algorithm for multi-tenancy resource calculations with applied constraints

$$\begin{aligned} C_{0i}(N+\Delta N,T+\Delta T) &= C_{0i}(N+\Delta N) + C_{0i}(T+\Delta T); \\ M_{0i}(N+\Delta N,T+\Delta T) &= M_{0i}(N+\Delta N) + M_0 + M_{0i}(T+\Delta T);\ i \in I\ . \end{aligned} \tag{12}$$

The second part is to calculate resource demand vectors due to each of tenants $T+\Delta T$ with users $N+\Delta N$ in an active instance i with each k of a set of applied constraints K,

such as response time and transaction rate. Then, the maximum resource demand vectors are selected from $k = 1$ *to* K and $t = 1$ *to* $T + \Delta T$:

$$\begin{aligned} C_{max,i}(N+\Delta N,T+\Delta T) &= \text{Max}\{C_{0i}(N+\Delta N,T+\Delta T),C_{ki}(t)\};\ \forall\ k \in K\ . \\ M_{max,i}(N+\Delta N,T+\Delta T) &= \text{Max}\{M_{0i}(N+\Delta N,T+\Delta T),M_{ki}(t)\};\ \forall\ t \in T+\Delta T\ . \end{aligned} \tag{13}$$

where $C_{ki}(t)$ and $M_{ki}(t)$ are resource demand vectors of CPU and storage due to constraint k on tenant t of instance i. The third part is to re-calculate a set of resource demand vectors for each i of shared instances I with tenants $T+\Delta T$ and users $N+\Delta N$. Then, the resource demand for a server s is calculated by sum over $i = 1$ *to* I and $a = 1$ *to* A on each s of available servers S. The residual resource of a server s is given by Equations (14):

$$\begin{aligned} C_{residual}(s) &= C_{initial}(s)\ - \textstyle\sum_{ia} C_{max,i}(N+\Delta N,T+\Delta T);\ s \in S\ . \\ M_{residual}(s) &= M_{initial}(s) - \textstyle\sum_{ia} M_{max,i}(N+\Delta N,T+\Delta T)\ ;\ \forall\ i \in I,\ \forall\ a \in A\ . \end{aligned} \tag{14}$$

where $C_{initial}(s)$ and $M_{initial}(s)$ are the initial resource of CPU and storage in a server s, respectively. $C_{residual}(s)$ and $M_{residual}(s)$ are the residual resource in CPU and storage in a server s, respectively if additional tenants ΔT_i and users ΔN_i are deployed in its shared instance i of an application a. The initial source must meet or exceed the total resource demand of a server s for each j of all resource types J. The effective residual resource score $E_{residual}(s)$ for all resource types J in a specific server s can then be calculated using several different methods. In our multi-tenant resource placement tool, a total score over all resource types J with their weighting factors w_j between *0.0* and *1.0* is used. From equations (14):

$$\begin{aligned} E_{residual}(s) &= w_{j=1} * C_{residual}(s) + w_{j=2} * M_{residual}(s) + \ldots\ ;\ s \in S\ . \\ \textstyle\sum_j w_j &= 1.0;\ \forall\ j \in J\ . \end{aligned} \tag{15}$$

Priority rules can be used to set weighting factors w_j of resource type j in the order of its importance and contributions to the effective residual resource score in the list of resource types J. Finally, specific server s^* with minimum effective residual resource score is selected for deployment of additional tenants ΔT_i and users ΔN_i in its shared instance i of an application a.

6 The Multi-tenant Placement Model

The placement of multiple applications in a set of available servers with optimization is illustrated in Case 1 of Figure 5. There are six available servers, namely S1, S2, S3, S4, S5 and S6 with different initial resources and five applications, namely A1, A2, A3, A4 and A5. Four instances of the same application A1, namely I1, I2, I3 and I4, have been deployed on S1, S2 and S4. Instances of applications A2, A3, A4 and A5 have also been deployed on S2, S3 and S5. The principle rule of optimization in the placement is to deploy a new instance on the server with the smallest residual resource left after meeting the resource requirement of this new instance. As a result, larger chunks of residual resource will be retained in other servers for later use by an

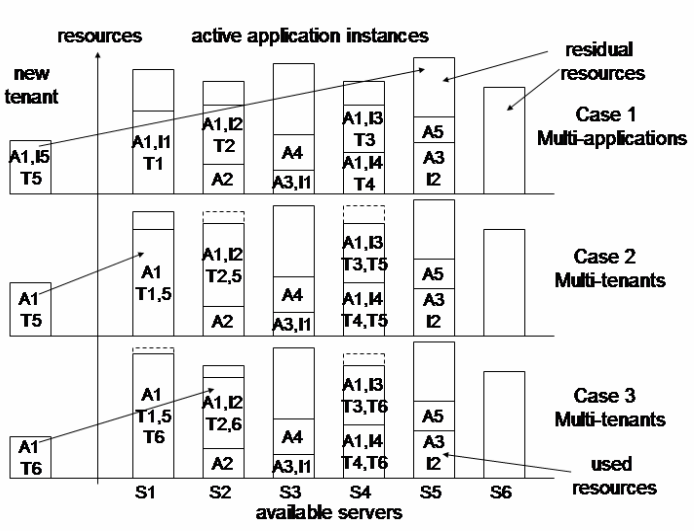

Fig. 5. Comparison of our new multi-tenant placement model with other previously reported placement models for multiple applications

application instance with a higher resource demand. First, let us assume that these application instances in Case 1 only support multi-users but not multi-tenants. Thus, a new instance I5 of A1 has to be created and deployed for a new tenant T5 even though there are existing instances I1, I2, I3 and I4 of the same application A1 running in S1, S2 and S4. Obviously, servers that have large enough residual resource to meet the resource requirement of I5 are S3, S5 and S6. With optimization, the traditional placement methods [12-14] or commercial products [16,17] will deploy I5 on S5 to leave larger chunks of residual resource on S3 and S6.

Now, let us assume that all these applications also support multi-tenants in addition to multi-users. Once again, traditional placement methods [12-14] or commercial products [16,17] with optimization will still deploy I5 on S5. However, the placement result using our new multi-tenant placement model is very different. As illustrated in Case 2, we may not need to create a new instance I5 of A1 for a new tenant T5 because there are existing instances I1, I2, I3 and I4 of the same application A1 running on S1, S2 and S4. First, we need to test whether the residual resource in one of servers S1, S2 and S4 would be large enough to meet the expanded resource requirement of I1, I2, I3 or I4 with an additional new tenant T5. As shown in Case 2, the expanded resource requirement of I1 with two tenants T1 and T5 will be within the resource limit on S1 while that of I2 with two tenants T2 and T5 will exceed the resource limit on S2. The expanded resource requirement of either I3 or I4 with two tenants T3 and T5 or T4 and T5 will also exceed the resource limit on S4. Obviously, our multi-tenant placement model with optimization will deploy the new tenant T5 into the instance I1 as the second tenant without creating another application instance I5. Case 3 illustrates the placement of another new tenant T6 for an application A1. Once again, the expanded resource requirement of I1 with three tenants T1, T5 and T6 will exceed the resource limit on S1 while that of I2 with two tenants T2 and T6 will be

within the resource limit on S2 this time because the resource requirement for T6 is smaller than that of T5. The expanded resource requirement of either I3 or I4 with two tenants T3 and T6 or T4 and T6 will also exceed the resource limit on S4. Instead of creating a new instance I6, this new tenant T6 for an application A1 will be placed on an existing instance I2 as the second tenant.

7 The Framework and Algorithm of a Multi-tenant Placement Tool

In an Internet data center, multiple SaaS offerings of application instances are active on shared physical resources, such as CPU and storage, of a set of computing servers in a distributed computing environment. When new tenants subscribe to a new SaaS offering, these new tenants need to be assigned to new or specific active instances under constraints due to SLA specifications of all tenants. Any server devoted to a new offering must have the required capacity of computing resource to host a new instance or an active instance with additional tenants and users without compromising SLA requirements of all tenants. Moreover, security restrictions on tenants in a shared instance cannot be violated. However, traditional application placement tools are not tenant aware [16,17]. Their approaches primarily focus on static or dynamic placement of applications to available servers based on their physical resources with or without load balance or rebalance. In these placement tools, a new application instance is always created and deployed for a new tenant. They cannot assign a new tenant into an active instance because they do not know how much extra resource requirements of an active instance with additional tenants and users.

An architectural framework of our multi-tenant placement tool is shown as block diagrams in Figure 6. This framework provides capabilities of multi-tenant resource calculations with applied constraints and the placement of tenants and instances for

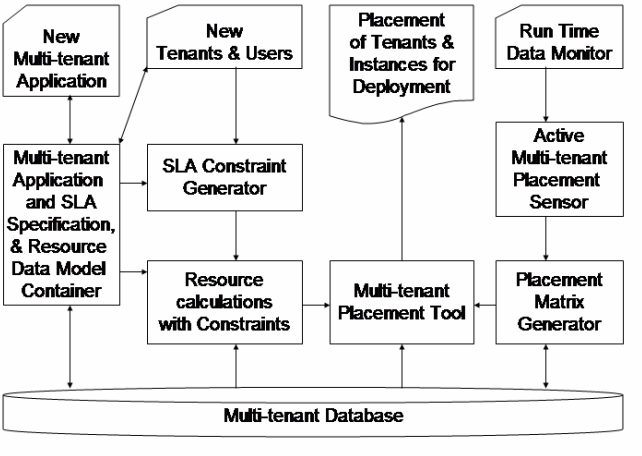

Fig. 6. An architecture framework of a multi-tenancy placement tool

multi-tenant application deployment. It consists of one output and three input modules, six essential functional modules and a multi-tenant database. The flow diagrams in Figure 6 also show the logical flows of information among modules and database. The "New Multi-tenant Application" module provides graphical user interface (GUI) and scripts for an administrator to input specifications and multi-tenant data models of a new software application or to modify existing ones. The "New Tenants and Users" module provides GUI for an administrator to enter numbers of new tenants and users in each tenant, and select the application required for deployment. The administrator also enters the SLA specification for each new tenant. The "Multi-tenant Application and SLA specification, & Data Model Container" module holds, stores, retrieves and deliveries these multi-tenant data from and to other modules and the multi-tenant database. The "Run Time Data Monitor" module constantly monitors and collects resource usage profile of each active instance in each server. It also provides information on performance parameters and utilization rate of each server. The "Multi-tenant Active Placement Sensor" calculates resource usages of each active instance and residual resource of each server. The "Placement Matrix Generator" constructs and stores resource usage and residual matrices. The dimension of these matrices is two with J x I, where J is the number of resource types and I is the number of instances in a server. The initial resource matrix $\mathbf{O_{initial}(j,s)}$ and residual resource matrix $\mathbf{O_{residual}(j,s)}$ of resource type j and server s are then constructed based on information from the active placement sensor. They are used in Equations 15 of Section 5. The "SLA Constraint Generator" module constructs and stores constraints due to SLA requirements for new tenants, and retrieves constraints for active tenants on a shared instance in a specific server s. The "Resource Calculations with Constraints" module

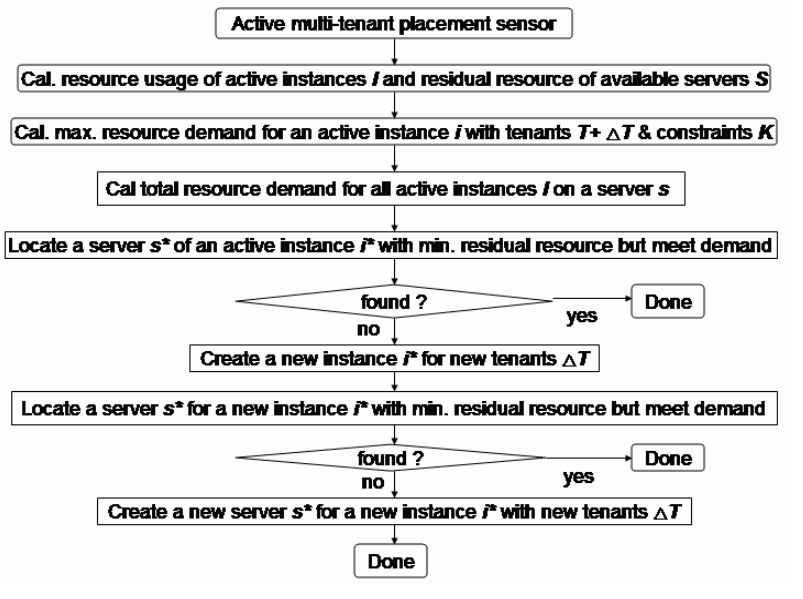

Fig. 7. A multi-tenancy placement algorithm

calculates required physical resources, such as CPU, memory, storage and network bandwidth, with applied constraints for new and active tenants in a shared instance on a specific server as described in Sections 3, 4 and 5. This is to make sure that SLA requirements are meet for all tenants in a shared instance. Moreover, cross tenant security restrictions are not violated, such as prohibition of tenant T1 and T2 deployed in the same application instance I1 and/or on the same server S1.

The "Multi-tenant Placement Tool" module constructs a package for placements of new tenants on specific instances and/or new instances on specific servers for multi-tenant application deployment according to our proposed multi-tenant placement model described in Section 6. The flow chart of a multi-tenant placement algorithm is shown in Figure 7. First, resource usage of all instances I, initial and residual resources of all available servers S are calculated based on information from the active placement sensor. Second, the maximum resource demand for each shared instance i with tenants $T+\Delta T$ and constraints K are calculated as described in Section 5. Third, the total resource demand for each server s with its shared instance i of tenants $T+\Delta T$ and its other instances $I-1$ are calculated. Fourth, a particular server s^* with its particular instance i^* of tenants $T+\Delta T$ is located because its residual resource is the minimum among all servers s, and within its resource limit. Fifth, a new instance i^* is created for new tenants ΔT if a particular s^* is not found. Sixth, a particular server s^* with the new instance i^* is located because its residual resource is the minimum among all servers s with the same new instance i^*, and within its resource limit. Finally, a new instance i^* is created in a new server s^* for new tenants ΔT if a particular s^* is still not found.

8 Implementation and Industrial Experiences

Most features and functions of the multi-tenant placement tool described in this paper have been implemented in Java. A generic sorting algorithm has been used to match the demand list from high to low with the residual source list from low to high. The residual source list is sorted once again after each match or placement. Optimizations based on the placement of one, two or three tenants at a time have been investigated. This multi-tenant placement tool is being integrated within the provisioning subsystem of an IBM internal project. Since the tool is integrated in a fully automated end-to-end provisioning, it saves time for administrators, increases their efficiency and productivity in managing the placement of tenants and instances for multi-tenant application deployment by simplifying and automating the placement processes. Its performance with two applied constraints, CPU and storage, as a function of new tenants or servers is shown in Figure 8. The number of new tenants equals the number of servers while the number of active tenants equals to half the number of servers. The CPU spent on the placement algorithm is found to depend on the number of new and active tenants, and the number of servers. Our preliminary data based just on one set of data has indicated that the CPU spent on the placement increases linearly with the number of new tenants or servers up to 100 tenants or servers. Results from more data sets will be collected in the future. Our preliminary results have confirmed the stability and usefulness of this multi-tenant placement tool. This tool has shown to minimize the number of servers deployed, resulting in maximum cost savings, in a

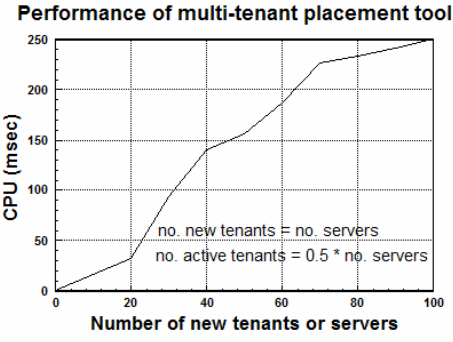

Fig. 8. The performance of multi-tenant placement tool

distributed computing environment. It has also shown to meet constraints of all tenants, and provide higher service reliability and increase uptime.

For our industrial experiences, we have found that it is very time consuming and tedious to measure tenant or user specific resource data for a new multi-tenant application. It will be useful if we can calculate new resource data based on available resource data of other active multi-tenant applications using their activity correlation functions. We have also found that it is hard to accurate project memory requirement for multiple tenants in a shared instance. As a result, an estimated upper limit on memory requirement is often used. Our preliminary results have revealed that several crucial constraints, such as response time and transaction rate, always play important roles in determining multi-tenant resource requirements. They have also indicated that limiting factors or bottlenecks in multi-tenant applications depend on its computing activities, such as Web, transaction, computing and database. Moreover, users of this multi-tenant placement tool have request additional functions, such as adding an additional placement rule to assign each server with a minimum load, merging several instances of the same application in a server into one, and migrating tenants among instances. They would also like to use this placement tool to tell whether a specific application would benefit from the multi-tenant deployment.

9 Conclusion and Discussion

In this paper, we have outlined calculations of resource requirements for multiple tenants in a shared application instance with applied constraints using our hypothetical multi-tenant resource data models. We have also described novel methods for the optimal placement of tenants and instances based on our proposed multi-tenant placement model without violating any SLA requirements of all tenants in a set of servers. We have architected and implemented the first of a kind, a multi-tenant placement tool for application deployment using a minimum number of servers, and thus with maximum cost savings in a distributed computing environment. However, other challenges, such as database security and data isolation, tenant view filter and data encryption, remain un-tackled. In the future, these challenges should be addressed with proved solutions.

Characteristics of multi-tenancy in four different software layers: application, middleware, VM and operating system, are important factors in studying and understanding limiting factors or bottlenecks in multi-tenant SaaS applications. Based on our intuitions, the resource sharing and cost savings are relatively high for multiple tenants in a shared instance while the security isolation is high and performance impact is low for multiple instances or VMs per server. In the future, this multi-tenant placement tool can be used to verify these multi-tenant characteristics once multi-tenant resource usage data on these four different layers have been measured.

Acknowledgments. The authors would like to thank A. Karve for his work on the integration of this multi-tenant placement tool with other IBM internal projects. The authors would also like to thank J. Batstone for her support.

References

1. Iod: Sotware as a Service, Director Publications Ltd., London (2002)
2. Iyar, S.: Why Buy the Cow, Santa Clara (2007)
3. Kobilsky, N.: SAP CRM on-demand, SAP Forum (2006)
4. Gianforte, G.: Multiple-Tenancy Hosted Applications: The Death and Rebirth of the Software Industry. RightNow Technologies Inc. (2005), http://wwww.rightnow.com
5. Chong, F., Gianpaolo, C., Wolter, R.: Multi-Tenant Data Architecture, Microsoft Corporation (2006), http://www.msdn2.microsoft.com/
6. Fisher, S.: The Architecture of the Apex Platform, salesforce.com's Platform for Building On-Demand Applications. In: Proc. of the 29th IEEE Int'l Conference on Software Engineering, p. 3. IEEE Press, New York (2007)
7. Guo, J.G., Sun, W., Huang, Y., Wang, Z.H., Gao, B.: A Framework for Native Multi-Tenancy Application Development and Management. In: Proc. of the 9th IEEE Int'l Conference on E-Commerce Technology, pp. 551–558. IEEE Press, New York (2007)
8. Kwok, T., Nguyen, T., Lam, L.: A Software as a Service with Multi-Tenancy Support for an Electronic Contract Application. In: Proc. of IEEE Int'l Conference on Services Computing, pp. 28–33. IEEE Press, New York (2008)
9. Mendoza, A.: Utility Computing Technologies, Standards, and Strategies. Artech House Publishers, Norwood (2007)
10. Herroelen, W., Reyck, B.D., Demeulemeester, E.: Resource-Constrained Project Scheduling: A Survey of Recent Developments. Computers and Operations Research 25(4), 279–302 (1998)
11. Brucker, P., Drexel, A., Mohring, R., Neumann, K., Pesch, E.: Resource-Constrained Project Scheduling: Notation, Classification, Models and Methods. European Journal of Operational Research 112, 3–41 (1999)
12. Hartmann, S., Kolisch, R.: Experimental Evaluation of State-of-the-Art Heuristics for Resource-Constrained Project Scheduling Problem. European Journal of Operational Research 127, 394–407 (2000)
13. Kolisch, R.: Efficient Priority Rules for the Resource-Constrained Project Scheduling Problem. Journal of Operations Management 14, 179–192 (1996)
14. Bouleimen, K., Lecocq, H.: A New Efficient Simulated Annealing Algorithm for the Resource-Constrained Project Scheduling Problem and its Multiple Mode Version. European Journal of Operational Research 149, 268–281 (2003)

15. Rajkumar, R., Lee, C., Lehoczky, J., Siewiorek, D.: A Resource Allocation Model for QoS Management. In: Proc. of the 18th IEEE Real-Time Systems Symposium, pp. 298–307. IEEE Press, New York (1997)
16. Islam, N., Prodromidis, A., Squillante, M., Fong, L., Gopal, A.: Extensible Resource Management for Cluster Computing. In: Proc. of the 17th Int'l Conference on Distributed Computing Systems, pp. 561–568. IEEE Press, New York (1997)
17. Bagchi, S., Hung, E., Iyengar, A., Vogl, N., Wadia, N.: Capacity Planning Tools for Web and Grid Environments. In: Proc. of the 1st Int'l Conference on Performance Evaluation Methodologies and Tools, pp. 25–34. ACM Press, New York (2006)
18. Bhatti, M.A.: Practical Optimization Methods. Springer, New York (2000)