Journal of Universal Computer Science, vol. 12, no. 2 (2006), 160-186
submitted: 12/1/05, accepted: 15/7/05, appeared: 28/2/06 © J.UCS

POCA : A User Distributions Algorithm in Enterprise Systems with Clustering

Ping-Yu Hsu
Department of Business Administration, National Central University, Taiwan
pyhsu@mgt.ncu.edu.tw

Ping-Ho Ting
Department of Hospitality Management, Tunghai University, Taiwan
ding@thu.edu.tw

Abstract: As enterprises worldwide race to improve real-time management to improve productivity, customer services and flexibility, huge resources have been invested into enterprise systems (ESs). All modern ESs adopt an n-tier client-server architecture, which includes several application servers to hold users and applications. As in any other multi-server environment, the load distributions, and user distributions in particular, become a critical issue in tuning system performance.

In stateful ESs, a user who logs onto an application server and stays connected to the server for an entire working session, which can last for days, evokes each application. Therefore, admitting a user onto an application server affects not only current but also future performance of that server. Although the n-tier architecture may involve web servers, there is little in the literature in Distributed Web Server Architectures that considers the effects of distributing users instead of individual requests to servers.

The algorithm proposed in this paper gives specific suggestions in user distributions and the minimal number of servers required based on the application reusability threshold. A heuristic version of the algorithm is also presented to improve the performance. The paper also discusses how to apply association rules to predict new user behavior when distributing users in the run-time. The distributions recommended by the algorithms are compared against the Round-Robin distributions on a set of real data derived from a mid size company. The result shows that the user distributions suggested have better performance than Round Robin distributions.

Keywords: Clustering, User Distribution, Profile, Load Balancing, Enterprise Systems
Category: H.3.4 H.2.8 H.3.3 H.2.4 H.1.2 K.6.4

1 Introduction

As enterprises world-wide strive to reduce time and costs spent in dealing with customer enquires and request, large amount of resources have been invested into developing, purchasing, and implementing Enterprise Resource Planning (ERP), Customer Relationship Management (CRM), Supply Chain Management (SCM), and all sorts of other enterprise systems (ESs). Due to different design objectives, ESs have various functionalities, but all modern ESs share a common IT foundation, namely, the n-tier client-server architecture. This architecture has

a database server in the storage layer, multiple application servers in the service layer, several web servers in the interface layer and browsers or other access devices in the presentation layer. Figure 1 shows a sample n-tier architecture. In this style of architecture large number of programs are held and executed within the application servers. When users log onto a system, they either select an application server or they are being assigned to one by the system.

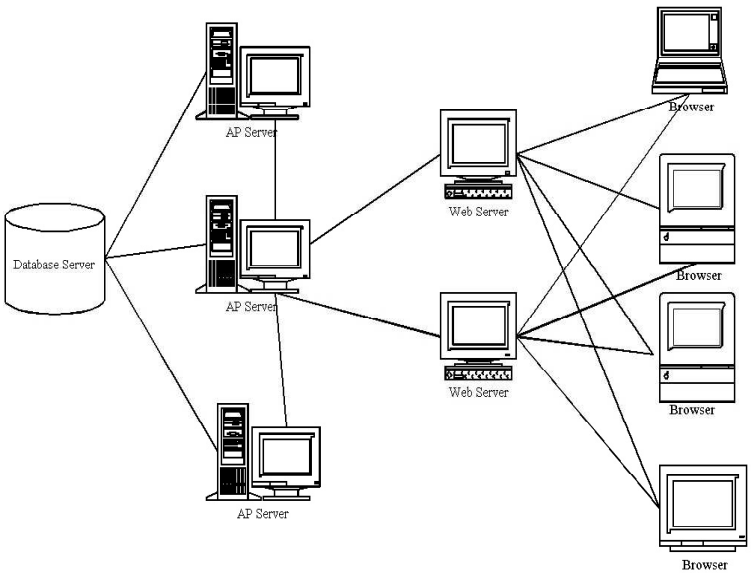

Figure 1: The N-Tier Client-Server Architecture

As in any OLTP system, ES users have low tolerance toward slow system responses. If a system responds to data entries or queries too slowly, users often lose patience and complain about the lack of suitable service. Yet, the number of ES users is growing for most enterprises as the number of business processes incorporated into ESs has increases dramatically. Therefore, keeping response time under control is a vital issue for most system administrators.

When memory shortage or CPU speed dampen down a server response, two approaches, namely, scale-up and scale-out, can improve the performance. With the scale-up approach, more memories or CPUs are installed on the same server to alleviate the constraints. With the scale out approach, a new server is installed to reduce the loads on the original servers and thus improve the performance. In general, the former approach costs less money and requires less effort to maintain the systems. However, enterprises may take the latter approach for the reason

of high availability, hardware limitation, or lack of hardware compatibility. An extra server can reduce the risk of system shut down if one of the servers fails and thus increases the system availability. All servers have limitations on the sizes of memories and numbers of CPUs that can be installed on a machine and no new components can be installed when the limitation is met. Since computer hardware phased out quickly, organizations may adversely find that no compatible memories and CPUs can be installed with the existing hardware.

When an ES has multiple application servers, distributing users with similar behaviors to the same application servers plays an important role in tuning system performance[1]. In a stateful system, each user's requests during a logging-on session are directed to the same application server. In enterprise systems, a logging on session may last for an entire working day. Therefore, user assignments have long-term impact on system performances for stateful application servers comparing to stateless application servers, which may assign each request to a different server.

The most important piece of the algorithm is the POCA(Profile Oriented Clustering Algorithm) algorithm, which partitions users into groups within the patterns of accessed transaction sets. The algorithm is novel in finding all clusters satisfying a given threshold instead of finding a number of locally optimized clusters. The cluster algorithm is also special in partitioning data made by sets of categorical data, instead of data points marked with Manhattan distances. A heuristic version of POCA is also proposed to improve the speed of user clusterings.

The research procedure is shown in figure 2. The procedure starts with collecting user profiles from an enterprise system. The profile is consisted of a set of transactions accessed by users. The transactions that are accessed frequently are labeled as regular transactions in the second step. The frequencies are compared against profile support threshold and user support threshold. The profile support threshold is used to screen transactions that are seldom used by all users and user support threshold is to find transactions which are accessed frequently by each single user. The regular transactions are further analyzed to form associated regular transaction in the third step with confidence threshold. Associated regular transactions are designed to predict the behavior of new and infrequent users who do not have enough records in the user profile. In the distribution phase, regular transactions are used to cluster users with two naval algorithms, namely POCA and HPOCA. The clusters formed by POCA require an extra step to form pick distributions which consists of the minimum number of clusters A hybrid distribution mechanism is also proposed to distribute users with or without patterns during the run-time.

[1] An application in an ES corresponds to an atomic and unbreakable transaction. In this paper, transactions and applications are used interchangeably.

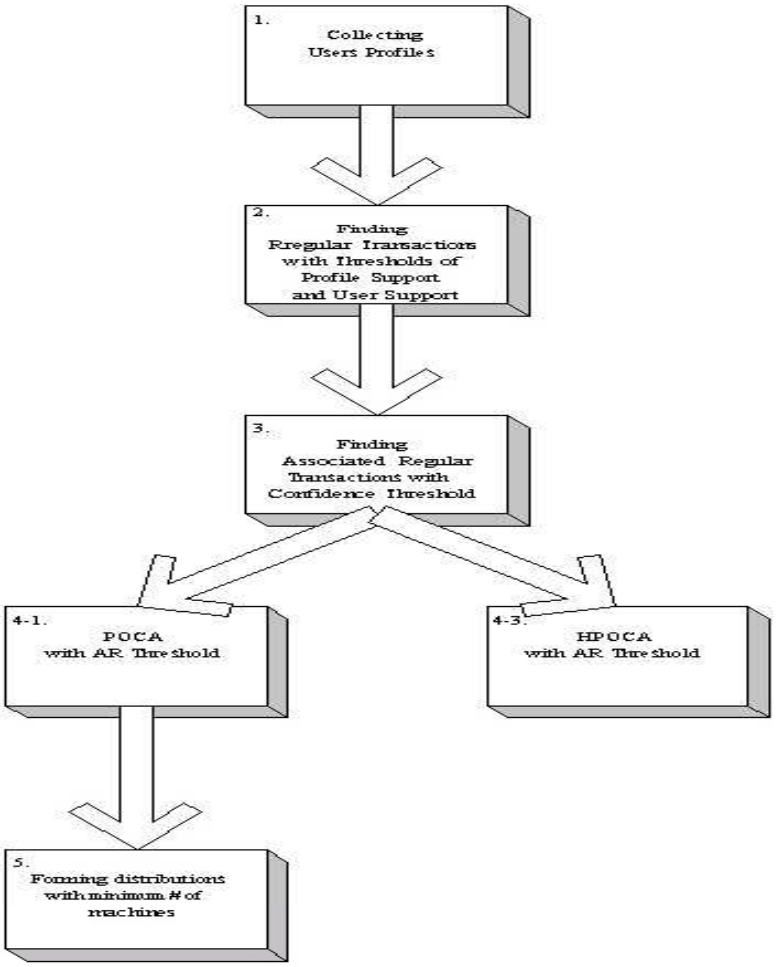

Figure 2: The Procedure of the Research

To verify the distribution results, a four-week user access logs were extracted from a mid size company. Three weeks of the data were used to form clusters and the forth week of the data were used to verify the effectiveness of the distribution. The effectiveness was measured against round-robin user distribution in terms of application buffer hitting ratios and entropies of user applications in each server.

To explain the algorithms and related procedures, the rest of the paper is organized into following sections. Applications are grouped into large itemsets with a traditional Apriori algorithm[11] to find frequent patterns. The process is explained in section 2. The association rules between single transactions and patterns are also discovered in the section. A group of users forms a cluster if

the group's Application Reusability (AR) exceeds a given threshold. AR is a similarity measure of user patterns grouped in the same set. The definition of AR and related properties are proved in section 3. The algorithm of POCA is explained in section 4. POCA computes all possible clusters and pick the ones with the properties of disjoined and comprehensive to form distributions. The distributions with the fewest number of clusters are returned to system administrators. POCA are proved ends in finite steps, and the distributions generated from POCA are complete and sound in section 5. A heuristic version of POCA are shown in section 6. A hybrid dispatching algorithm to distribute users in run-time is shown in section 7. The comparison between Heuristic POCA and Round-Robin based on real data in terms of buffer hitting ratio and application entropy are shown in section 8. A review of distributed web server architectures and clustering of categorial sets and other clustering technique are shown in section 9. Conclusion and possible extension of POCA are discussed in section 10.

2 Finding Users' Regular Transactions

To record system and user statuses, most enterprise systems include various tracing mechanisms. Among the various recordable data are user sessions and applications executed in session. We defined one day to be a session. We make up the transaction sets used by users per day to be a session. For the purpose of the paper, these data are transformed into user profiles. A user profile is a set of ⟨ user-id, transaction-set ⟩, where user-id is the account name of a user and transaction-set is the set of transactions accessed by the user in a session.

A sample user profile is shown in Table 2, which records the sessions of ten users. User 1, 3 and 6 have more than one sessions in the profile. User 1 access transaction A, B, E, F, and H in one session and A, B, E, and F in another session.

As careful readers may have found that the transactions accessed by user 10 in the profile shown in Table 2 is special because most of his/her transactions are unique and are not shared by others. Transaction G of users 2 in the first session is also unique. If the rarely used transactions are all stored in buffers, large sizes of buffers are needed and the utilization rates of these buffers are low. Therefore, only regularly accessed transactions are considered in AR. A user's regularly accessed transactions,termed as regular transactions, are transactions which occur in enough number of sessions in the corresponding user profile and are accessed often enough by the user.

Definition 1 *Given a user,u, a user profile, U, and a transaction, t, t is one of u's regular transaction in U if*

$$\frac{|\{s|t \in s.transaction\text{-}set, s \in U\}|}{|U|} \geq profile\ support\ threshold, and$$

User-Id	Transaction-Set
1	{A, B, E, F, H}
1	{A, B, E, F}
2	{A, B, E, F, G}
2	{A, B, E, H}
3	{A, B, E}
3	{B, E, F, H}
4	{I, J, K, L}
5	{B, I, J, K}
6	{B, I, J, L}
6	{B, I, J, K}
7	{O, P, Q, R}
8	{O, P, Q, R}
9	{P, Q, R, K}
10	{W, X, Y}

Table 1: User Profiles

$$\frac{|\{s|s \in U, s.user\text{-}id = u \wedge t \in s.transaction\text{-}set\}|}{|\{s|s \in U, s.user\text{-}id = u\}|} \geq user\ support\ threshold.$$

Profile support threshold and user support threshold are given by system administrators. The higher the threshold, the fewer the regular transactions users have.

To compute or estimate regular transactions for each user, three steps are employed. The first one computes large itemsets with any existing set oriented pattern discovering algorithm, such as [5, 13]. The large itemsets computed from the algorithms have supports higher than the profile support threshold in the associated user profile. In the second algorithm, each large 1-itemset is examined against each user to form users' regular transactions. For new users who do not have accumulated enough entries to computer personal regular transactions, the paper propose to predicate their regular transactions with the association rules computed with known algorithms.

Figure 3 shows the stages in computing regular transactions.

Any set oriented minding technology that can find large itemsets with supports higher than a given profile support threshold can be used. In the section, traditional Apriori-like algorithm [5] is example to explain the process of finding transaction patterns. The algorithm includes two phases: candidate generation and pruning phases. The candidates surviving the pruning phase are termed as large itemsets. Candidates are generated and pruned level by level. In the first level, each transaction in the user profile is treated as a candidate. Each candidate is checked against the user profile to count the number of transaction

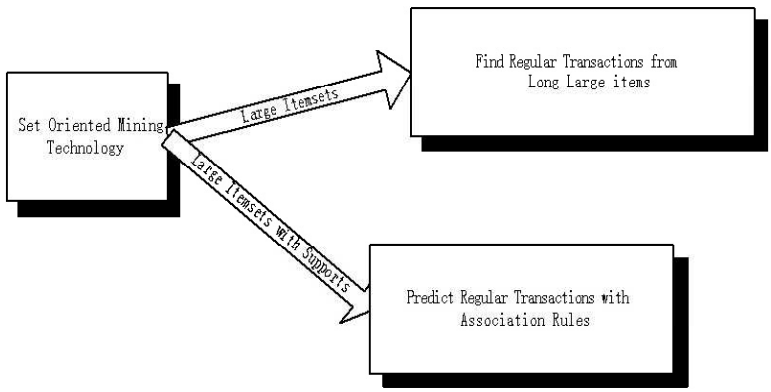

Figure 3: The Stages of Computing Regular Transactions

sets which include the candidate. Candidates with counts under a threshold are pruned and the remaining candidates are large itemsets. Large itemsets are then "joined" to form next-level candidates. The process continues until no new large itemsets are found.

If profile support threshold is set at 20%, the set of level 1 large itemsets of the sample user profile is {A, B, E, F, H, I, J, K, P, Q, R}; the level 2 set is {AB, AE, AF, BE, BF, BH, BI, BJ, EF, EH, IJ, IK, JK, PQ, PR, QR}; the level 3 set is {ABE, ABF, AEF, BEF, BEH, BIJ, IJK, PQR}; the level 4 set is {ABEF}. Therefore, the set of patterns generated from the Apriori-Like Algorithm is {A, B, E, F, H, I, J, K, P, Q, AB, AE, AF, BE, BF, BH, BI, BJ, EF, EH, IJ, IK, JK, PQ, PR, QR, ABE, ABF, AEF, BEF, BEH, BIJ, IJK, PQR, ABEF}.

The second step in computing users' regular transactions is to map transactions in large itemsets to users. A transaction is a user's regular transaction if it happens in enough number of the user's sessions. One obvious way to do so is taking every Level 1 large itemsets and check it against each users' transaction sets. The itemset is one of the user's regular transaction if the item occurs in enough number of the user's transaction sets.

Assume the user support threshold is set at 40%, the regular transactions of the running example is shown in Table 2.

New users do not have any records in the user profiles and do not have associated regular transactions. However, dispatching programs still need to dispatch them in run-time. Therefore, help for dispatching programs to guess the patterns of new users are in order.

If each new user provides one of the transactions she/he wishes to access after log on, the dispatching program can check if the transaction has high association with any large itemsets. If so, the union of the large itemsets denote the user's

User-Id	Regular Transactions
1	{A, B, E, F, H}
2	{A, B, E, F, H}
3	{A, B, E, F, H}
4	{I, J, K}
5	{B, I, J, K}
6	{B, I, J, K}
7	{P, Q, R}
8	{P, Q, R}
9	{P, Q, R}
10	∅

Table 2: Regular Transactions

Predicted Regular Transaction set.

Definition 2 *The Associated Regular Transactions of a transaction, t, under a set of large itemsets, P, a user profile, U, is*

$$AT(t) = \Sigma_{p \in P \ \wedge \ t \in p} CP_U(p|t) \geq \textit{confidence threshold},$$

where $CP_U(p|t) = \frac{|\{s|s \in U, p \in s.\textit{transaction set}\}|}{|\{s|s \in U, t \in s.\textit{transaction set}\}|}$

By setting the confidence threshold at 80%, the Associated Regular Transactions of transactions in large 1-itemsets in the running example is shown in Table 2.

Since the algorithms needed to find the Associated Regular Transactions are trivial when large itemsets are ready. The paper does not include the algorithm either.

3 The Definitions of Similarity Measure, Clusters, and Distributions

Load balancing programs utilizes the benefit of multiple servers at the cost of wasting memories in keeping duplicated programs and data. In sophisticated application servers with hundreds of people on-line, the memory needed for transactions are considerable [2]. Therefore, users with similar regular transactions should be grouped into one cluster, which are then assigned to a server[1, 14]. This section defines the measure of similarity and proves related properties. Formal definitions of clusters and distributions are also included.

Transaction	PT	Confidence
A	ABE	100%
B	AB	100%
E	ABE	83%
F	BEF	100%
H	BEH	100%
J	IJK	100%
K	IJK	100%
P	PQR	100%
Q	PQR	100%
R	PQR	100%

Table 3: Associated Regular Transactions with Confidence Threshold at 80%

Definition 3 *A Cluster is a set of users that share similar regular transactions.*

The similarity of users in a cluster is measured by AR, Application Reusability. The AR of a transaction in a cluster is defined as the percentage of users in the cluster evoking the transaction, and the AR of a cluster is defined as the average AR of transactions in the cluster.

Definition 4
- *The Regular transactions, T, of a user, u are defined as T(u)*
- *AR of a transaction, t, in a cluster, c, is defined as*

$$\boldsymbol{R}(t,c) = \frac{|\{u \,|\, u \in c \; and \; t \in \boldsymbol{T}(u)\}|}{|c|}$$

.

- *The AR of a cluster, c, is defined as the average AR of regular transactions in the cluster.*

$$\boldsymbol{R}(c) = \frac{\sum_{t \in \boldsymbol{T}(u) \; and \; u \in c} \boldsymbol{R}(t,c)}{|\{t \,|\, u \in c \; and \; c \in \boldsymbol{T}(u)\}| * |c|}$$

.

With |c| as one of the denominators, AR implicitly favors cluster with few number of users

Lemma 1 *Every entry in an AR has a value between 0 and 1.*

Proof
ARs are positive and therefore are always greater than or equal to 0. AR of a transaction, t, in a cluster, c, is

$$\begin{aligned}\boldsymbol{R}(t,c) &= \frac{|\{u \mid u \in c \text{ and } t \in \boldsymbol{T}(u)\}|}{|c|} \\ &\leq \frac{\{u \mid u \in c\}}{|c|} \\ &\leq 1\end{aligned}$$

Cluster AR is the averages of Transaction AR and therefore also has values between 0 and 1.

□

Lemma 2 *Let a cluster, c, has the AR of* $\frac{p}{u*q}$ *where p is the times of regular transactions being accessed by users in the cluster, u is the number of users in the cluster, and q is the number of different regular transactions in the cluster. If a new user is added to the cluster with n regular transaction accesses and m of them are different from the existing transactions then the new AR is* $\frac{p+n}{(u+1)*(q+m)}$.

Theorem 1 Conditional Anti-Monotonicity of AR $\boldsymbol{R}(c)$ *decreases with new user added to c, if the number of transactions accessed by the new user is fewer than or equal to the average number of transactions accessed by original users.*

Proof
if c has an AR of $\frac{p}{u*q}$ *where u is the number of users in the server, q is the number of transactions in the cluster, and p is the times of transactions being accessed by users in cluster c. A new user is added to the server with n transaction accesses; m of them are new transactions. , the new AR should be* $\frac{p+n}{(u+1)*(q+m)}$.

$$\begin{aligned}\frac{p}{u*q} - \frac{p+n}{(u+1)*(q+m)} &= \frac{p*(u+1)*(q+m) - (p+n)*u*q}{u*q*(u+1)*(q+m)} \\ &= \frac{p*u*m + p*q + p*m - n*u*q}{u*q*(u+1)*(q+m)} \\ &= \frac{m*(p/q + p/uq) + (p/u - n)}{(u+1)*(q+m)}\end{aligned}$$

Therefore, $\frac{p}{u*q} - \frac{p+n}{(u+1)*(q+m)} \geq 0$ *if* $p/u \geq n$. □

Therefore, $\boldsymbol{R}$(c) has the property of Conditional Anti- Monotonicity, which allows POCA to prune hapless user groups.

Definition 5 – *A cluster whose AR exceeds a given ARThreshold is called qualified cluster.*

– *A set of clusters is comprehensive under a user profile, U, if the union of the clusters includes all and only all users with regular transactions.*

– *A set of clusters is disjoined if the intersection of any two clusters in the set is empty.*

– *A set of qualified clusters is a distribution under a user profile, U, if they are comprehensive under U and disjoined.*

The clusters of {1, 2, 3}, {4, 5, 6}, and {7, 8, 9} have ARs of 100%, 11/12, and 100%, respectively, under the running example. If the ARthreshold is set at 55% then all three are qualified clusters. The three clusters are both comprehensive and disjoined in the example, and therefore form a valid distribution.

4 Clustering and Distributing by POCA

POCA returns distributions that satisfy administrator constraints and has the fewest number of clusters, and the rules associating single transactions to predicted regular transactions. The constraints include an AR threshold, min-support, rule confidence, and coverage threshold. The recommendations guarantee that when all frequent users logging on the system and accessing all regular transactions, each server still has an AR above the given ARThreshold (AR Threshold). Information included in the recommendations are number of servers, ARs, needed units of user, transaction, and data buffers.

POCA relies $\geq_U$, which is a chain, to hold Conditional Anti-Monotonicity and to form each user combination at most once.

Definition 6 *Let S be the set of users in a user profile, U. The order* $\geq_U$ *is defined on S such that for any* $u_1, u_2 \in S$, $u_1 \geq_U u_2$ *if*

– $\boldsymbol{T}(u_1) > \boldsymbol{T}(u_2)$, *or*

– $\boldsymbol{T}(u_1) = \boldsymbol{T}(u_2)$ *and user-id of* $u_1 \leq$ *user-id of* u_2.

Theorem 2 $\geq_U$ *is a chain.*

Proof
For all $x, y, z \in$ *user set of* U,

– $x \geq_U x$, *since the user-id of x are the same,*

- *$x \geq_U y$ and $y \geq_U x$ imply that x and y have the same user-id and therefore $x = y$,*
- *$x \geq_U y$ and $y \geq_U z$ imply one of the four cases,*
  1. *$\boldsymbol{T}(x) > \boldsymbol{T}(y)$ and $\boldsymbol{T}(y) > \boldsymbol{T}(z)$, imply $\boldsymbol{T}(x) > \boldsymbol{T}(z)$ and therefore $x \geq_U z$,*
  2. *$\boldsymbol{T}(x) > \boldsymbol{T}(y)$ and $\boldsymbol{T}(y) = \boldsymbol{T}(z)$, imply $\boldsymbol{T}(x) > \boldsymbol{T}(z)$ and therefore $x \geq_U z$,*
  3. *$\boldsymbol{T}(x) = \boldsymbol{T}(y)$ and $\boldsymbol{T}(y) > \boldsymbol{T}(z)$, imply $\boldsymbol{T}(x) > \boldsymbol{T}(z)$ and therefore $x \geq_U z$,*
  4. *$\boldsymbol{T}(x) = \boldsymbol{T}(y)$ and $\boldsymbol{T}(y) = \boldsymbol{T}(z)$ and user-id of $x \leq$ user-id of y and user-id of $y \leq$ user-id of z imply user-id of $x \leq$ user-id of z and therefore $x \geq_U z$,*
- *$x \geq y$ or $y \geq x$ since even if x and y have the same value of $\boldsymbol{T}$, they have different user-ids, which are comparable.*

Therefore, $\geq_U$ is a chain.

□

POCA includes two major steps in computing the recommendations – computing the set of qualified clusters and selecting clusters to form server distribution. The main steps are listed as following:

Initialization: for each user with regular transactions, turns the user into a single-user cluster. These clusters form C_1, the 1-user cluster set.

Composing C_{i+1} from C_i: Conditional join C_i with C_1 to form C_{i+1}. A cluster c_i from C_i is added by one user in c_1 from C_1 if two criteria are met. The first one states that the user from c_1 has lower rank in $\geq_U$ than any user in c_i. The second criterion asserts that the new cluster has an AR value exceeding the given threshold.

Repeating Last Step Until No New Clusters are Generated: If C_{i+1} is empty then POCA has found all qualified clusters in C_1, ..., and C_i; Otherwise, POCA has to repeat the last step.

Selecting Clusters to Form Distributions: Finding the fewest number of qualified clusters to form distributions. The algorithm includes a loop to check if i clusters can form a distribution where $1 \leq i \leq C_1$. The loop is aborted when distributions are found.

In the running example, the first cluster set is formed by turning each user into a cluster. The result of $C1$ is {{1}, {2},{3},{4},{5},{6},{7},{8},{9}}

If setting the AR threshold at 55%, C_2, the set of 2-user clusters, is equal to the join of C_1 and C_1. C_2 = {{1, 2}, {1, 3}, {2, 3}, {5, 6}, {5, 4}, {6, 4} {7, 8}, {7, 9}, {8, 9} }. C_3 is the conditional join of C_2 and C_1. C_3 = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}}. C_4 is the conditional join of C_3 and C_1. C_4 and associated ARs are listed in Table 4. C_5 is empty since potential 5-user clusters have ARs lower than 55%.

Four-Item Cluster	AR
{1, 2, 3, 7}	18/32
{1, 2, 3, 8}	18/32
{1, 2, 3, 9}	18/32

Table 4: The 4-User Cluster Set

Now that all cluster sets are ready, it is time to select clusters to form user distributions.

The selection continues by examining 1-cluster distribution, 2-cluster distribution, etc. In the running example, 1-cluster and 2-cluster distributions are empty since there are no 9-user and 5-user clusters. On the other hand, 3-cluster distributions have various alternatives and are all returned to system administrators. 3-cluster distributions and corresponding ARs are listed Table 5. The performance issues of set unions and intersections have been intensively studied by many researches and is not the main topic of the paper. POCA just picks one to examine the comprehensiveness of clusters.

User Distributions	ARs
{1, 2, 3}, {4, 5, 6}, {7, 8, 9}	100%, 11/12, 100%
{1, 2, 3, 7}, {4, 5, 6}, {8, 9}	18/32, 11/12, 100%
{1, 2, 3, 8}, {4, 5, 6}, {7, 9}	18/32, 11/12, 100%
{1, 2, 3, 9}, {4, 5, 6}, {7, 8}	18/32, 11/12, 100%

Table 5: 3-Cluster Distributions and ARs

The algorithm of POCA is listed in Figure 4. The algorithm returns all the distributions that satisfy the requirements and let system administrators to decide which distribution he/she prefers.

Input: U ♯User Profile,
MinProfileSupport ♯Profile Support Threshold,
MinUserSupport ♯User Support Threshold ,
ConfidenceThreshold ♯ Threshold of Associating Transactions to Large Itemsets ,
ARThreshold ♯Threshold of Application Reusability,
Output: UD (User Distributions)
PR (Association Rules of Predicting Patterns from Transactions)

P = Apriori(U, MinProfileSupport) ♯ finding patterns
T = FindRegular((P, U, MinUserSupport) ♯ Definition 1
PR = FindPredictedTransactions(P,ConfidenceThreshold,U) ♯ Definition 2

♯Transforming a set of users with regular transactions to C_1
$C_1 = \{\{u.\text{User-id}\} \mid u \in \text{U and } \boldsymbol{T}(\text{u.User-id}) \neq \emptyset\}$
♯compute C_{i+1} from conditional join of C_i and C_1
For i = 1; $C_i \neq \emptyset$; i++ {
  $C_{i+1} = \emptyset$
  For each $c \in C_i$
    For each $u_1 \in \cup C_1$ {
      If $\forall u \in c, u \geq_U u_1$ then
        If $\boldsymbol{R}(c \cup \{u_1\}) \geq$ ARThreshold then
          $C_{i+1} += (c \cup \{u\})$ }}
♯Building a Cluster Set
$C = \emptyset$
For i= 1; $C_i \neq \emptyset$; i++
  $C = C \cup C_i$

♯ Combining clusters to form distribution
UD = $\emptyset$
For i = 1; i $\leq |C_1|$ and UD = $\emptyset$; i ++ {
  For each combination of $c_1, c_2, \ldots, c_i \in C$
    If $\sum(|c_1|, \ldots, |c_i|) = |C_1|$ and $\cup(c_1, \ldots, c_i) = \cup C_1$ then
      UD $\cup=$ $\{\langle c_1, \ldots, c_i\rangle, \langle |c_1|, \ldots, |c_i|\rangle,$
        $\langle |\cup \boldsymbol{T}(c_1)|, \ldots, |\cup \boldsymbol{T}(c_i)|\rangle,$

Return UD, PR
End Algorithm

Figure 4: Profile Oriented Clustering Algorithm

5 The Correctness of POCA

In this section, we prove that POCA ends in finite steps, and the distributions generated from POCA are complete and sound. That is, all distributions satisfy system administrators' requirements are generated and all generated distributions are correct. By requirements, we mean that all distributions are composed of the fewest number of clusters, clusters are comprehensive and disjoined, and each cluster has an AR above the designated threshold.

Theorem 3 *POCA stops in finite steps.*

Proof
The proof of termination is straightforward. C_{i+1} *generated in the cluster generation loop is a set of (i+1)-user clusters. Each loop increases one to i. In the worst case, the loop iterates* $|C_1|$ *times and produces the set of* $C_{|C_1|+1}$*, which must be empty. The empty set leads to the termination of the cluster generation loop.*

□

Theorem 4 *POCA returns only sound distributions.*

Proof
The proof of soundness is trivial since the selection criteria in the cluster generation loop finds only qualified clusters with AR exceeding ARThreshold. The distribution generation loop starts from 1 to find the fewest number of clusters needed to form a distribution. The if-condition in the loop asserts that clusters in distributions must be comprehensive and disjoined. Therefore, distributions returned must be correct.

□

Theorem 5 *POCA returns all correct distributions.*

Proof
The distribution generation loop produces all combinations of clusters which are comprehensive and disjoined and include the fewest clusters. Therefore, the completeness of POCA lies on the completeness of clusters generated in the cluster generation loop.

For each cluster c_i*, which has i users and* $i > 1$*,* c_i *can be seperated into two subsets* c_{i-1} *and* c_1 *such at the former subset has* $i - 1$ *users and the latter has 1 user and* $\forall u \in c_{i-1}, u_1 \in c_1, u \geq_U u_1$*, since* $\geq_U$ *is a chain.*

$$\forall u \in c_{i-1}, u_1 \in c_1, , c = c_{i-1} \cup c_1, u \geq_U u_1 \Rightarrow \boldsymbol{T}(u) \geq \boldsymbol{T}(u_1) \Rightarrow Avg(\boldsymbol{T}(u)) \geq \boldsymbol{T}(u_1) \Rightarrow \boldsymbol{R}(c_{i-1}) \geq \boldsymbol{R}(c_i)$$

Therefore, for each cluster with more than one user, the AR of the sub-cluster that excludes the least element in the cluster is higher than or equal to the AR of the complete cluster. With the same argument, if a cluster has an AR lower than the ARThreshold, all of its super-clusters formed by adding users with ranks lower than any user in the cluster also have ARs lower than the ARThreshold. Therefore, POCA generates complete clusters with ARs above ARThreshold.

□

6 Performance Improvement of POCA

In section 4, POCA potentially examines all possible combination of users with regular transactions in the associated user profile. If the number of users is k, the complexity of POCA is O(k^k) in the worst case. By incorporating Chain into POCA, the complexity in the worst case can be reduced to O(2^k). In this section, we propose a Heuristic POCA, namely HPOCA to further reduces the complexity.

The HPOCA uses a greedy method to distribute users. HPOCA tries to squeeze as many users as possible into a cluster by finding the user that can reduce the least amount of AR into a cluster. The main approach is listed as following:

Initialization: Sorting users with regular patterns into a list, $\boldsymbol{L}$, in the descending order of numbers of transactions. Taking the first user from $\boldsymbol{L}$ and form the first cluster, C_1.

Composing C_i from $\boldsymbol{L}$:

while $\boldsymbol{L}$ is not empty
  Retrieve a user,u, from $\boldsymbol{L}$ such that $\boldsymbol{R}(C_i \cup \{u\}$ has the greatest value.
  If $\boldsymbol{R}(C_i \cup \{u\}) \geq$ ARThreshold then add u to C_i and delete u from $\boldsymbol{L}$.
  Else
    i = i +1
    $C_i = \boldsymbol{L}(1)$ and delete $\boldsymbol{L}(1)$.
  End If
End While

The clusters found by HPOCA constitute a distribution.

7 An AR Based Hybrid Dispatching Approach

Each ES typically has a dispatching program listening to networks and accepts user requests. The program resides an application server, intercepts user requests, and direct them to application servers.

Assuming the system administrator in our running example picks the distribution of {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}}. The case of user 1, 2, 4, 7, and 8 have logged on and user 5 and 6 are waiting in the web server is depicted in Figure 5.

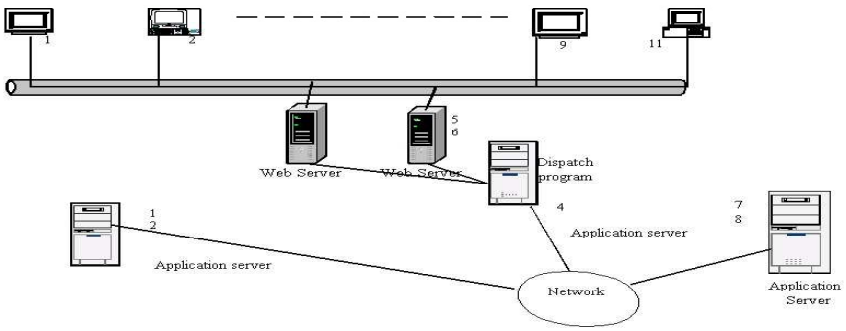

Figure 5: Users are Distributed through a Dispatching Program

The distributions suggested by POCA bases on frequent patterns in user profiles. For new and infrequent users, POCA does not suggest their distributions directly but returns association rules, PR (Prediction Rules), in the output to help dispatching program make the decision.To apply the rules, a new user only needs to provide a transaction he/she plan to evoke after logging on the ES. With the association rules, a dispatching program can distribute a user according to its associated predicted regulation transactions. If the first transaction does not lead to any predicted regular transactions, then the single transaction works as the basis for dispatching.

An AR Based Hybrid dispatching algorithm distributes users while keeping the AR of each server as high as possible. In the dispatching procedure, users

are distributed to a server by one of the three alternatives:

- If a regular user logs on, then send the user to the recommended server and return to listening mode.
- If an infrequent user logs on with a transaction, then find the predicted regular transactions implied by the transaction. If no entry matched then the single transaction is treated as the predicted regular transaction.
- Compute the potential new AR in each server with the addition of the user with predicted regular transactions. Assign the user to the server with the highest AR, and update the AR in the corresponding server.

The distribution in the running example has ARs of 100%, 11/12, and 100% in the three servers. If a new user with user-id 10 wishes to log on the system and submits an A as the first transaction then the user has a assumed predicted regular transaction of ABE, according to Table 2. The ARs after adding ABE to the three servers would be 18/20, 14/24, 12/24. Therefore, the new user is distributed to the first server, and the distribution becomes {1, 2, 3, 10}, {4, 5, 6}, {7, 8, 9}.

8 Simulation and Comparison

Several experiments are conducted on real data collected from a mid size machinery company based in Taichun, Taiwan. The company has their SAP system up and running since 2002. Five weeks of user access logs are extracted from the system to perform the experiment. Four weeks of the data are used to suggest distributions. The fifth week of data is used to evaluate the quality of the suggested distributions. The qualities of the user distributions of HPOCA are benchmarked against the qualities of user distributions based on Round-Robin distributions, with which users are distributed to servers according to the logging-on orders.

The qualities of distributions are measured by Application Hit Ratios and Entropies. The Application Hitting Ratio of a server is defined as the number of transaction accesses hitting a stored version of the transactions in the memory over the total number of transactions accessed in the server. The Application Hit Ratio of a distribution is the average Application Hit Ratios of servers suggested by the distribution. When an application is accessed in a server and yet does not have a stored version, the transaction is stored into the memory buffer. When memory buffers are full, the slot with the eldest application is emptied to store the latest application. The entropy of a server is defined as $-\sum p_i \log_2(p_i)$, where p_i is the probability of transaction i being accessed by users in the cluster. By computing the entropy of each server, we can measure the similarity of regular patterns of users allocated to the server.

The Experiment is implemented on Matlab 6.1 and executed on a Pentium 4-1.8 GHz Microsoft XP Server system with 256 Megabytes of main memory. With the HPOCA, distributions can be suggested within one minute.

In the experiment, 1,853,689 access logs are collected, with which 56 users are found to have regular patterns when the user support and profile support thresholds are set at 0.3 and 0.1, respectively. The average number of different transactions in regular patterns is 7.7.

Five experiments are performed by setting AR thresholds at 0.6, 0.5, 0.4, 0.3, and to 0.2, respectively. The number of servers suggested by HPOCA in each case is shown in Figure 6. The suggested numbers of machines are 5, 4, 3, 2, and 1 for the five AR thresholds.

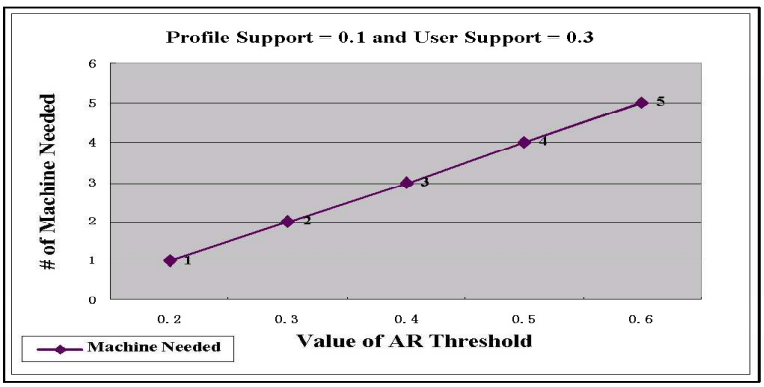

Figure 6: Numbers of Servers Suggested with HPOCA

The hitting ratios of the five user distributions are further analyzed against the sizes of memory buffers. In each distribution, hitting ratios against buffers with sizes ranging from 15 to 28 are compared. The hitting ratios of distributions suggested by HPOCA under various memory sizes are compared against the result derived from Round Robin Distribution. The comparisons are shown in Figure 7, 8, 9, 10. The Figures show that HPOCA has better performance than Round-Robin in each case.

The qualities of distributions suggested by HPOCA and Round-Robin are also contrasted in the dimension of entropies. The comparison is shown in Figure 11 Readers can again find that HPOCA beats Round-Robin in all five cases.

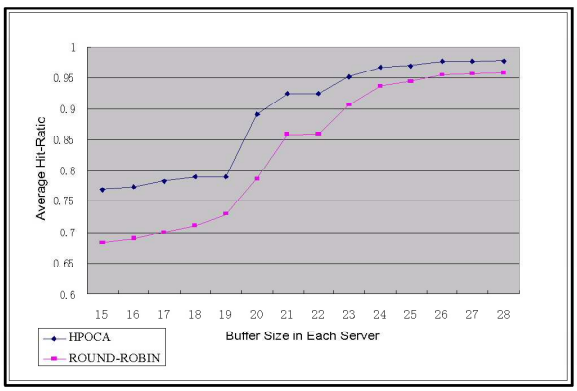

Figure 7: Hitting Ratios of Distributions of HPOCA and Round Robin in Two Server Scenario

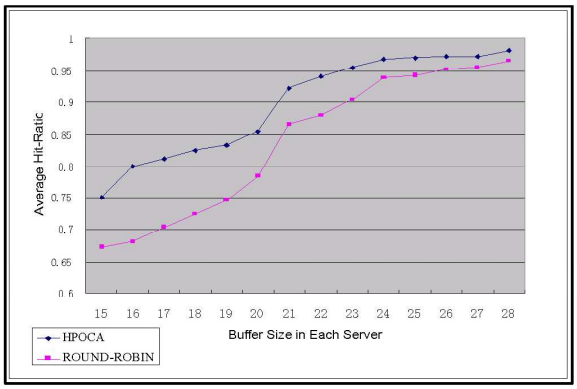

Figure 8: Hitting Ratios of Distributions of HPOCA and Round Robin in Three Server Scenario

9 Related Work

9.1 Web load distribution

With the Internet rush, many researches have been devoted to distributing user requests with Distributed Web Server Architecture to improve the performance of web servers. Depending on the locations where request distributions happen, these researches are classified into client-based, DNS (Domain Name Server)-based, dispatcher-based, and server-based by [24, 7, 6]. The client-based architecture distributes requests from client sites and requires the browser software to know the location of distributed servers. This approach adds work loads to

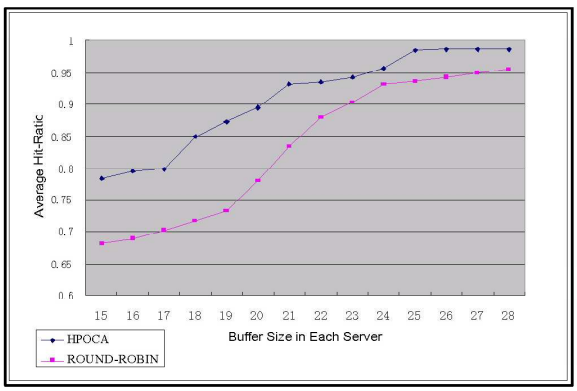

Figure 9: Hitting Ratios of Distributions of HPOCA and Round Robin in Four Server Scenario

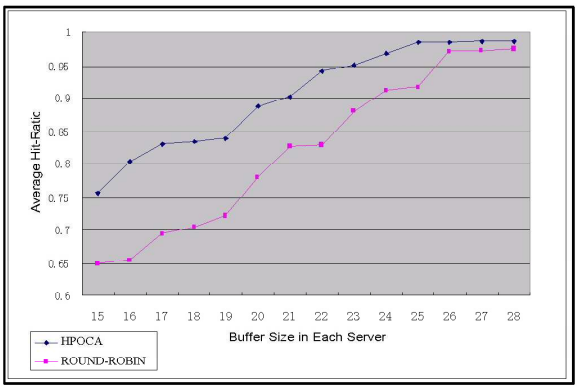

Figure 10: Hitting Ratios of Distributions of HPOCA and Round Robin in Five Server Scenario

browser software and may even increase network traffic when browsers querying for distributed locations of web servers. The DNS-based architecture translates symbolic site names into IP addresses according to predefined algorithms, such as Round Robin approach. The architecture may not always get the desired results since some other DNSs on the net may cache a fouled version of the translations. Dispatcher-based architecture employs a central dispatchers as a gateway to distributed web servers. All requests are directed to the central server that may itself becomes a bottleneck in a heavily loaded network. The server-based architecture cures the shortfalls of dispatcher-based architecture by having all web servers process request distributions. Each server has the knowledge of loads and content in all other servers and dispatches requests accordingly. The archi-

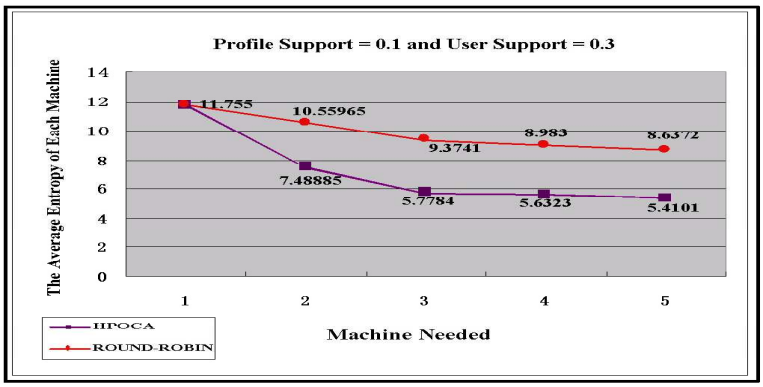

Figure 11: Entropies of Distributions Suggested by HPOCA and Round Robin

tecture requires extra work and loading in the web servers. The dispatcher- and server-based architectures can be combined to form a two-level architecture in which requests are distributed to a cluster of web servers by the dispatcher and the requests are further routed by servers in the cluster [27].

Two kinds of granularity are proposed to distribute data in the architecture, namely, basic component of web pages and sets of web pages. Instead of storing complete web pages in distributed servers, [3, 19, 20] propose to split web pages into several weblets that are components of web pages and can be executed independently to generate data in pages. When generating web pages, only weblets need to be recomputed, static parts of pages can be fetched from caches directly. With careful distributing of weblets, systems can guarantees the QoS (quality of service) of page generation. Ng et al. argue that distributed web servers do not need to duplicate entire content in different servers since only a small portion of web pages are requested in each session [21]. Web pages regularly queried in a session are grouped into a migrating unit that is distributed to a web server. The dispatch is accomplished in a two-level distributed web server architecture. The central dispatcher distributes a session to a server according to the first requested page and the loads in related servers. Every subsequent request is directed to the designated server until a requested page is not held in the server. In this case, the server redirects the requests to another server that holds the page.

9.2 Clustering

Since current Http protocol is stateless, each request is routed independently to a web server[6]. All of the above researches assume that requests can be independently route to different servers, where as in the application servers of

ESs, requests from the same users have to be routed to the same server. POCA is a clustering algorithm, it is also related to clustering techniques. Clustering algorithms partition data points into clusters so that the clusters exhibit good characteristics, such as high similarities, short inter-distances, etc. Clustering literatures are classified into partitioning clusterings and hierarchical clusterings [17, 9, 12]. If k clusters are needed, partitioning Clustering choose k centroids initially and gradually tune the constituents of each clusters or centroids with some criteria function until a locally optimized characteristic is met. Hierarchical clusterings can be further divided into agglomerative and divisive clusterings. As the name suggested, agglomerative clusterings gradually merge smaller clusters into larger clusters until k clusters are found. Divisive clustering, on the other hand, splits larger clusters into smaller clusters until k clusters are found. POCA is more close to agglomerative although it does not have predefined cluster numbers.

Many data type arising from cluster applications can be modeled as bipartite graph[8, 18, 16] examples include users and ranked items in collaborative recommendation systems[3, 4], gene and patient in clustering of microarray[22, 26, 15, 23, 25]. Bipartite clusters treat two types of objects connected with a weighted bipartite graph as the first class citizens and try to find the clusters that group objects from the two types in such a way that the summation of the weights of the links connecting objects in different clusters are minimum. The user allocation problem tackled by POCA treats users as the first class citizens, transactions as properties of users and the number of servers needed as the number of clusters. Hence, clusters of bipartite graphs cannot be directly applied to solve the issues.

Most clustering algorithms employ Euclidean or Manhattan distances to compute similarity. The shorter the distances the similar the data points in the clusters are. However, Euclidean distances are not ideal for clustering categorical data. For example, to cluster transaction sets with Euclidean distances, each set has to be translated into a sparse binary vector. In the running example, the second session of user 1, {A, B, E, F} is translated into $\langle 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\rangle$. The huge number of zeros can easily skew the distances between transaction sets. For example, a transaction set of {A} is translated into $\langle 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\rangle$ and {I} is translated into $\langle 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0\}\rangle$. Since {A} and {I} have a distance of $\sqrt{2}$, and {A} and {A, B, E, F} have a distance of $\sqrt{3}$, the former pairs has shorter distance than the latter. The conclusion violates the general perception of set operations. Therefore, Euclidean distances are not ideal for clustering categorical data.

Many set oriented algorithms use Jaccard coefficient [17] to compute distances. Given two sets T_1 and T_2, their Jaccard coefficient is $\frac{T_1 \cap T_2}{T_1 \cup T_2}$. However, Jaccard coefficient along cannot describe the number of elements in each cluster,

which are important to calculate the buffer efficiency. Another major work in clustering categorical data is ROCK [10], which proposes to cluster transaction sets based on links between nodes, which are composed by common neighbors between any pair of nodes. A common neighbor of two transaction sets is a transaction set sharing similar items with the two sets. ROCK puts two elements into the same cluster if the count of common neighbors exceed certain threshold. ROCK also has the same drawbacks as Jaccard coefficient. Hence, common categorial clustering technology is not suitable for clustering users in the application.

POCA is an agglomerative clustering algorithm that finds all clusters with characteristic exceeding a given threshold and returns distributions with the fewest of number of clusters. The reason for such a unique approach is that in current status, some factors affecting system performances are difficult to model and thus the optimized solutions provided by ordinary algorithms may not best fit system administrators' need. Therefore, providing a set of good enough suggestions and related data may suit system administrators more.

10 Conclusion

Managers in enterprises often add users to ESs as they extend E-business practices to various parts of corporate operations. With the addition of each user, new pressures on performances are brought upon the systems. Yet, system response time is one of the most important factors in measuring user satisfactions.

To boost performance and increase system availability, organizations may employ several application servers. With multiple application servers in the scene, distributing users with similar application requirements to the same application servers can increase buffer utilization and improve system performance. Since user sessions in an stateful ESs tend to cover entire working days, user profiles should include all applications needed in typical working days.

To provide suggestions for the number of servers needed and distributing users among the application servers, an algorithm named POCA is designed. By taking user specified Application Reusability Threshold, POCA returns the distributions with the least number of servers needed.

The procedure of POCA roots its theory on Application Reusability (AR), which is applied in two levels – application and cluster level. The AR of an application measures the probability of the application being accessed by users in the same cluster. The AR of a cluster is the average ARs of all applications in the cluster. A cluster with high AR values means users in the cluster share similar applications. Therefore, POCA asks system administrators setting a threshold to screen clusters without enough ARs.

Cluster level ARs have the property of Conditional Anti-Monotonicity, which states that if the number of the transactions of a new user added to a cluster is

fewer than (or equal to) the average number of transactions accessed by existing users then the AR of the new cluster is lower than (or equal to) the AR of the original cluster. With the property, POCA prunes hopeless search branches and stops the iterations when an empty cluster set is found.

By finding the least number of servers needed, POCA has the complexity of 2^k where k is the number of users with regular patterns. To speed up the computation, the algorithm of HPOCA is also presented to heuristically derive the distribution.

Although frequent users and regular transactions are stable in ESs, new users are added to the systems from time to time. These users have no entries in user profiles and are distributed by a hybrid dispatching program that distributes frequent users according to a selected distribution and new users with dynamic ARs. A transaction of the new user is checked to find its predicted regular transactions. If an entry is found, the dispatching program associating the user with the predicated transactions, otherwise, the single transaction is associate with the user. The associated transaction is then used to compute which server has the highest AR after accepting the user. The user is assigned to the server with the highest new AR.

To measure the quality of distributions suggested by HPOCA several experiments are conducted based on real data extracted from an ES system of a mid size company. The qualities are measured by Application Hit ratios and Entropies of applications of users in the same servers. The quality of distributions suggested by HPOCA are compared against Round Robin user distributions. HOPCA's results are better than that of Round Robin's in both measurement.

Traditional load balance algorithms are classified into dynamic and static. The former decides request distributions with static property of system configurations; the latter take into account current system situation in distributing users. POCA extends the time line into future by considering a user's regular requirement at the point of user logs on.

Several issues require further studies, such as modeling user profiles with sequences, dynamically updating user patterns, incorporating CPU and systems loads into dispatching and distribution algorithms, and improving the efficiency of POCA. Another extension is to investigate the possibility of building cluster groups of servers. In the scenario, a logical cluster is supported by a disjoined set of servers. In the run time, a dynamic load balance is activated to distribute users logically allocated to one cluster by POCA to servers in the same group based on the run time loading in the servers.

Acknowledgements

This study is supported by National Science Council, Taiwan, Republic of China, through the Project No.92-2416-H-008-006. We would like to thank anonymous

referees for their invaluable comments on this work.

References

1. SAP AG. *System R/3 Technicale Consultant Training 1 - administration*, chapter R/3 WorkLoad Distribution. SAP AG, 1998.
2. SAP AG. *System R/3 Technicale Consultant Training 3 - Perf. Tuning*, chapter R/3 Memory Management. SAP AG, 1998.
3. Woo Hyun Ahn, Woo Jin Kim, and Daeyson Park. Content-aware cooperative caching for cluster-based. *The Journal of system and software*, 69(1):75–86, 2004.
4. Y.Chen andJ.Wang and R.krovetz. Content-based image retrieval by clustering. In *In Proceedings of the 5th ACM SIGMM international workshop on Multimedia information retrieval*, pages 193–200, 2003.
5. R. Argawal and R. Srikant. Fast algorithms for mining associations rules. In *Proceedings of International Conference in Very Large Data Bases*, pages 487–499, 1994.
6. H. Bryhni, E. Klovning, and O. Kure. A comparison of load balancing techniques for scalable web servers. *IEEE Network*, 14:58–64, 2000.
7. V. Cardellini, M. Colajanni, and P.S. Yu. Dynamic load balancing on web-server systems. *IEEE Internet Computing*, 3:28–39, 1999.
8. I. Dhillon. Co-clustering documents and words using bipartite spectral graph partitioning. In *In Proceedings of the 7th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 269–274, 2001.
9. R. O. Duda and P. E. Hard. *Pattern Classification and Scene Analysis*. Wiley-Interscience Publication, 1973.
10. S. Guha, R. Rastogi, and K. Shim. Rock: A robust clustering algorithm for categorical attributes. *Information Systems*, 25(5):345–366, 2000.
11. J. Han and M. Kamber. *Data Mining: Concepts and Techniques*, chapter Mining association rules in large databases. Morgan Kaufmann Publisher, 2001.
12. J. Han and M. Kamber. *Data Mining: Concepts and Techniques*, chapter Clustering. Morgan Kaufmann Publisher, 2001.
13. J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proceedings of ACM-SIGMOD International Conference on Management of Data*, pages 1–12, 2000.
14. J.A. Hernándes. *The SAP R/3 Handbook*, chapter Distributing R/3 Systems. McGraw-Hill, 2 edition, 2000.
15. A. J. Holloway, R.K. van Larr, R.W. Tothill, and D.D. Bowtell. Options available - from start to finish- for obtaining data from dna microarrays ii. *Nature Genetics*, 32 (suppl):481–489, 2002.
16. H.Zha, X.He, C.Ding, H.Simon, and M.Gu. Bipartite graph partitioning and data clustering. In *In Proceedings of the 10th international conference on Information and Knowledge management*, pages 25–32, 2001.
17. A.K. Jain and R.C. Dubes. *Algorithms for Clustering Data*. Prentice Hall, 1988.
18. E. Jessup and D. Sorensen. A parallel algorithm for computing the singular value decomposition of a matrix. *Journal on Matrix Analysis and Applications*, 15(2):530–548, 1994.
19. P. Mohapatra and H. Chen. A framework for managing qos and improving performance of dynamic web content. In *Proceedings of Global Telecommunications Conference*, volume 4, pages 2460–2464, 2001.
20. S. Nadimpalli and S. Majumdar. Techniques for achieving high performance web servers. In *Proceedings of International Conference on Parallel Processing*, pages 233–241, 2000.

21. B. C-P. Ng and C-L. Wang. Document distribution algorithm for load balancing on an extensible web server architecture. In *Proceedings of International symposium on cluster computing and the Grid*, pages 140–147, 2001.
22. Jie Qin, Darrin P.lewis, and William Stafford Noble. Kernel hierarchical gene clustering from microarray expression data. *Bioinformatics*, 19(16):2097–2104, 2003.
23. J. Quackenbush. Microarray data normalization and transformation. *Nature Genetics*, 32(suppl):496–501, 2002.
24. Victor Safronov and Manish Parashar. Optimizing web servers using page rank prefetching for clustered accesses. *Information Sciences*, 150:165–176, 2003.
25. D.K Slonim. From patterns to pathways: gene expression data analysis comes of age. *Nature Genetics*, 32 (suppl):502–508, 2002.
26. Marcel Smid, Lambert C.J.Dorssers, and Guido Jenster. Venn mapping:clustering of heterologous microarray data based on the number of co-occurring differentially expressed genes. *Bioinformatics*, 19(16):2065–2071, 2003.
27. J. Zhang, T. Hamalainen, J. Joutsensalo, and K. Kaario. Qos-aware load balancing algorithm for globally distributed web systems. In *Proceedings of international conferences on Info-tech and Info-net*, volume 2, pages 60–65, 2001.