行政院國家科學委員會專題研究計畫　成果報告

以記憶體限制及模糊集群為基礎的大型企業系統使用者分配之研究

計畫類別：個別型計畫
計畫編號：NSC94-2416-H-029-010-
執行期間：94 年 08 月 01 日至 95 年 07 月 31 日
執行單位：東海大學餐旅管理系

計畫主持人：丁冰和

計畫參與人員：韋俊仲

報告類型：精簡報告
報告附件：出席國際會議研究心得報告及發表論文
處理方式：本計畫可公開查詢

中　華　民　國 95 年 10 月 30 日

行政院國家科學委員會補助專題研究計畫 ■ 成 果 報 告 □期中進度報告

以記憶體限制及模糊集群為基礎的大型企業系統使用者分配之研究

計畫類別：■ 個別型計畫　　□ 整合型計畫
計畫編號：NSC 94－2416－H－029－010－
執行期間：　94 年 08 月 01 日至 95 年 07 月 31 日

計畫主持人：丁冰和
共同主持人：韋俊仲
計畫參與人員：

成果報告類型(依經費核定清單規定繳交)：■精簡報告　□完整報告

本成果報告包括以下應繳交之附件：
□赴國外出差或研習心得報告一份
□赴大陸地區出差或研習心得報告一份
□出席國際學術會議心得報告及發表之論文各一份
□國際合作研究計畫國外研究報告書一份

處理方式：除產學合作研究計畫、提升產業技術及人才培育研究計畫、列管計畫及下列情形者外，得立即公開查詢
□涉及專利或其他智慧財產權，□一年□二年後可公開查詢

執行單位：東海大學餐旅管理系

中文摘要

隨著資訊科技的進步，企業面臨全球性的競爭，企業追求即時、快速及整合的資訊以利正確而快速的決策，大型的企業資訊系統如SAP ERP等就被需求著，這類的大型企業系統大都架構在N-tier的主從式架構底下，在這個架構底下就牽涉到使用者在應用伺服器的分配，這樣的分配法則，SAP等大型企業系統並無好的法則，只是依使用者的身份分配可能會使用的模組，這樣的結果使得企業系統的使用效率無法提昇，這類的問題屬於類別資料的使用者分配，國去所提出的演算法如Euclidean distances、Jaccard 係數並不適用在本研究上，過去文獻上並無更合適的文獻提出，本研究曾於九十三年有所討論與相關的發表，這次本研究所討論的是假設應用伺服器是在有限的記憶容量之下所設計的演算法，稱為HBC^{2}A。

HBC^{2}A演算法的特色是利用只用者的使用記錄並在有限的記憶體之下，進行使用者分配，這個演算法的特色使分配速度極快並能獲得合理的最佳解，本研究也利用台中某一家企業的資料進行模擬，模擬的結果以Hit-Ratio及Entropy等指標來看，顯示良好，值得推廣並應用到實務界，以利提升系統使用效率，並提昇競爭優勢。

Development of Users Distribution in Enterprise Systems with limited Buffer Size in Application Servers

Ping-Ho Ting[1], Kuan-Ching Li[2], and Chun Chung Wei[3]

[1] Dept. of Hospitality Management, TungHai University
Taichung 40704, Taiwan
ding@thu.edu.tw
[2] Dept. of Computer Science and Information Management, Providence University
Shalu, Taichung 43301, Taiwan
kuancli@pu.edu.tw
[3] Dept. of Information Management,ChungChou Institute of Technology
Yuanlin, Chunghwa 51003, Taiwan
ccwei@ms15.url.com.tw

Abstract. As enterprises worldwide race to improve their real-time management turnaround, which is essential requirement to improvements in productivity and service deployments, and therefore, large amount of resources have been invested into Enterprise Systems (ESs). All modern and robust ESs adopt a n-tier client-server architecture, which includes several application servers to hold users and applications. Currently, most web systems are stateless, which means that each request is routed independently to a different server at each time. However, for ESs, each request from same user is routed to the same application server.

Distributions in application and web servers are different in granularity. In the former scenario, a user represented by a set of transactions is the atomic element, while in the latter scenario, single request is the atomic element and different requests issued by the same user can be directed to different web servers. Until present time, few researches have been devoted in the user distribution to application servers in n-tier architecture.

In this paper, it is proposed a Heuristic Buffer Constraint Clustering Algorithm, namely HBC^2A , which is a Greedy-based strategy algorithm. The algorithm give suggestions of user distributions, the number of servers needed, and the similarity of user requests in each server. In addition, this algorithm is applied on a set of real data which is derived from the access log of an Enterprise System, in order to evaluate the quality of suggested distributions.

1 Introduction

For ESs that process daily business transactions, users typically have low tolerance on system performance. If a system responds to data entries or queries too slowly, users lose patience and complain loudly. Yet, the number of ES users keeps growing in most companies as the number of business processes incorporated into ESs increases. Therefore, keeping response time under control is a vital issue for most system administrators. To boost performances, activating more than one application servers become common practices in the industry.

When an ES has multiple application servers, distributing users to similar applications and application servers plays an important role in tuning overall system performance, as pointed out by documents of major ES systems [1, 13]. Throughout our text, an application in an ES corresponds to an atomic and unbreakable transaction, transactions and applications are used interchangeably.

In current practices, ESs do not automatically switch users to other application servers, due to the resources involved in the transmissions. As a user logs onto an application server, all related data such as authorizations, preferences, and created data are collected in the server's virtual memory, in order to create time-sharing working environment and reduce user effort in keying data. Besides, all applications executed by users are compiled and stored in memory. Data accessed by applications are also cached in memory to

improve the efficiency of systems. In many cases, the amount of data cached are huge, and as consequence, transferring a user to a different application server may trigger a transmission of huge amount of data. Thus, users are not switched automatically among servers in current practice.

In ESs, the dispatching mechanism needs to consider two criteria to gain reasonable performance: the number of users log on in each application server and the collection of applications executed in servers. Therefore, as each user consume hardware resources and the n-tier architecture has more than one application server, user distribution becomes one of the important issues in tuning ES performance [2]. In ESs, each application is evoked by a user who logs on an application server, and stays connected to the server for an entire working session, which can last for several hours and includes the execution of a set of applications. Therefore, admitting a user into an application server is equivalent to admitting a set of transactions into an application server, which marks a sharp difference between the distribution of application severs and traditional web servers.

In traditional web servers, requests are examined individually and those issued by the same user can be routed to different web servers. Commercial products, such as SAP R/3, equipped with a simple dispatching algorithm, considers only number of users and server response time. The task of grouping users is left to system administrators [1, 13]. In addition to the rough guideline of grouping financial users into one server and logistic users into another, system administrators need specific suggestions, such as explicit user distributions, the number of servers needed and the similarity of user requests in each server. To address the needs, this research paper proposes algorithm to suggest distribution based on user profiles. The distribution algorithm can the least number of servers needed that satisfy all the constraints of a system.

The scheme of the proposed research is shown in figure 1. The procedure is started with the collection of user profiles from an enterprise system. The profile is consisted of a set of transactions accessed by users in a given period of time. The transactions that are accessed frequently are labelled as regular transactions. The frequent accesses are compared against profile support threshold and user support threshold. The purpose of profile support is to screen transactions that are seldom used by all users and user support threshold is to find transactions which are accessed frequently by each 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 predicted the behavior of new and not frequent users, who do not have enough records in the user profile. In the distribution, regular transactions are used to cluster users with the novel algorithm prosed in this research paper, namely HBC^2A .

To explain the algorithms and related procedures, the rest of the paper is organized into the following sections. Applications are grouped into large itemsets with traditional Apriori algorithm[7, 11] to find frequent patterns. The process is explained in section 2. A group of users forms a cluster if the union of the users' transaction sets has an Application Match Ratio(AMR) exceeds a given threshold. AMR is a similarity measure of user patterns grouped in the same set. The definition of AMR and related properties are proved in section 3. An example of AMR based hybrid distributing approach is shown in section 4 Simulations with real data and comparisions with Round-Robin users distribution are shown in section 5. A review of distributed web server architectures and clustering of categorical sets is shown in section 6. Conclusion and possible extension of HBC^2A are discussed in section 7.

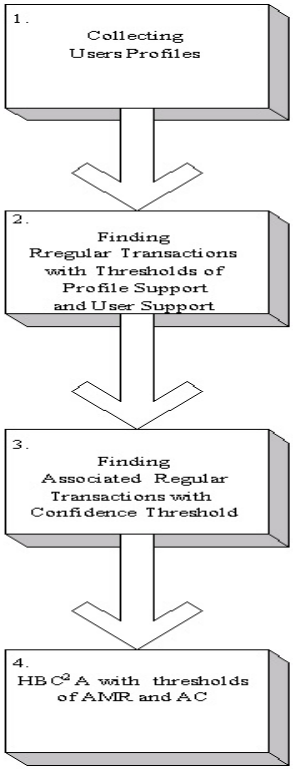

Fig. 1. Research scheme.

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 sessions. For the purpose of the paper, these data are transformed into user profiles. A user profile is a set of $\langle user - id, transaction - set \rangle$, 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 1, 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.

Table 1. User Profiles

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}

As careful readers may have found that the transactions accessed by user 10 in the profile shown in Table 1 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. 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*

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

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

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 [4, 14]. 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 2 shows the stages in computing regular transactions.

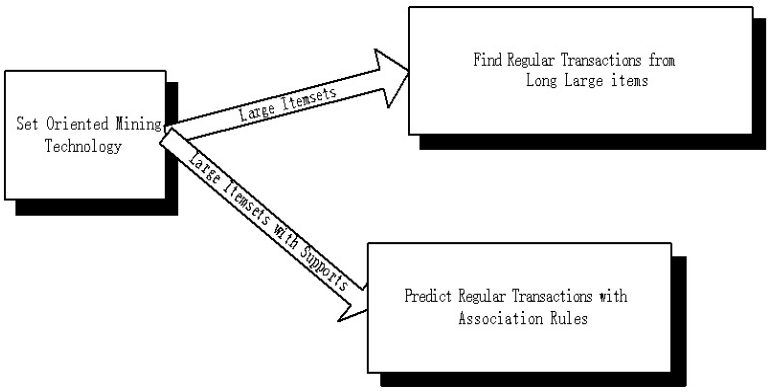

Fig. 2. The Stages of Computing Regular Transactions

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 the running example is shown in Table 2.

Table 2. Regular Transactions

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	∅

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 logging on, the dispatching program can check if the transaction has high association with any large itemsets. If so, the union of the large itemsets dentoe the user's 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*

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

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 3.

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 Clustering and Distributing by *HBC²A*

Systems with multiple servers gain performance speed at the cost of keeping duplicated programs and data in more than one servers. In sophisticated application servers with hundreds or thousands of users on-line all the time, the memory needed are considerable [2]. Therefore, users share similar transactions are grouped

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

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%

into one cluster, which is then assigned to an application server. This section proposes HBC^2A to cluster users and a straightforward algorithm to distribute clusters.

Definition 3 *A cluster is a set of users that share common applications in an Enterprise system.*

The quality of a cluster is measured by AMR , Application Match Ratio. The AMR of a cluster is defined as the ratio of AC versus the applications in the cluster, where AC denotes the number of applications that can be hosted in an application server without causing buffer swap. AMR is smaller than one when users in the cluster have more regular transactions than the buffers can hold. In this case, buffer swap occurs and the smaller the AMR is, the more the buffer swap will occur.

Definition 4 *The AC of an enterprise system is an integer number. The number denotes the number of applications that can resides in application servers of the enterprise systems without causing buffer swap.*

The regular transactions in a cluster are defined as the union of regular transactions of users grouped in the cluster.

Definition 5 *The number of regular transactions in a cluster, c, is defined as*

$$\|c\| = |\cup_{u \in c}\, u.regular\ transactions|$$

The AMR of a cluster, c, is defined as the ratio of AC to $\|c\|$. $AMR\ (c) = \frac{AC}{\|c\|}$.

Lemma 1 *The AMR of each cluster has a value between 0 and AC .*

Proof

AMR 's are positive and therefore are always greater than to 0.

Given a cluster, c

$$\begin{aligned} AMR\ (c) &= \frac{AC}{\|c\|} \\ &\leq \frac{AC}{1} \\ &\leq AC \end{aligned}$$

AMR of a cluster, therefore has values between 0 and AC .

□

Hence, system administrators can assign an AMR threshold between 0 and AC . By setting the threshold is between 0 and AC , the system administrators can tune the tolerance degree of buffer overflow.

Theorem 1 Anti-Monotonicity of AMR *AMR of a cluster decreases with the addition of any user with non-empty regular transaction set to the cluster.*

Proof

If a cluster, c, has the AMR of $\frac{AC}{p}$ *where p is the number of different transactions in the cluster. If a user with q new transactions is added to the cluster then the new AMR is* $\frac{AC}{p+q}$.

$$\begin{aligned}\frac{AC}{p} - \frac{AC}{p+q} &= \frac{AC * (p+q) - AC * p}{p * (p+q)} \\ &= \frac{AC * q}{p * (p+q)} \\ &\geq 0\end{aligned}$$

The case of $\frac{AC*q}{p*(p+q)} = 0$ *occurs when q=0, which means the regular transaction set of the new user does not contain any new transactions.*

□

Therefore, AMR has the property of Anti-Monotonicity, which means that adding a user to a cluster can only reduce the AMR of the cluster, unless the new transaction set does not contain any new transactions. The property can be used to prune hapless candidate clusters that have AMR under a threshold in the cluster forming algorithm, HBC^2A . In this paper, system administrators are requested to supply an AMR threshold. Candidate clusters with AMR smaller than the threshold are discarded.

Theorem 2 *The threshold of AMR must be smaller than or equal to* $\frac{AC}{|t_{max}|}$ *, where* t_{max} *is the largest regular transaction set in the user profile , to have all users grouped into at least one cluster.*

Proof
Any cluster c containing users with t_{max} *has AMR (c)* $\leq \frac{AC}{|t_{max}|}$. *If the threshold is larger than* $\frac{AC}{|t_{max}|}$, *then the users can not be included in any cluster.*

□

Definition 6

– *A qualified cluster is a cluster whose AMR exceeds a given threshold.*

- *A set of clusters is comprehensive under a user profile, U, if the union of the clusters includes all users with regular transactions in U.*
- *A set of clusters is disjoined if the intersections of any two clusters are empty.*
- *A set of qualified clusters is a distribution under a user profile, U, if they are comprehensive under U and disjoined.*

In the running example, if *AC* is set at 3, and *AMR* threshold at 0.5 , then the cluster of {1,2,3} ,{4,5,6} and {7,8,9} have *AMR* of 0.6, 0.75 and 1, respectively. The set composed by the three clusters is comprehensive, disjoined and forms a valid distribution. The running example is shown in Table 4.

Table 4. A set of qualified clusters when AC=3 and AMR=0.5

Qualified cluster	Users	Regular Transactions	AMR
Cluster 1	1,2,3	A,B,E,F,H	0.6
Cluster 2	4,5,6	B,I,J,K	0.75
Cluster 3	7,8,9	P,Q,R	1

We propose a Heuristic BC^2A , namely HBC^2A , HBC^2A returns distributions that satisfy constraints with the fewest number of clusters, and the rules associating single transactions to predicted regular transactions. The constraints include *AC* , an *AMR* threshold, profile support threshold, user support threshold, and rule confidence threshold. The recommendations guarantee that when all frequent users logging on the system and accessing all regular transactions, each server still has an *AMR* above the given *AMR* Threshold. Information included in the recommendations are number of servers, clusters of users, and *AMR*s of clusters.

The HBC^2A includes three steps in computing the recommendations - computing the set of qualified clusters and selecting clusters to form distribution. The main steps are listed as following:

Initialization: for each user with regular transactions, and these users form qure, Q. Sort Q on users by the number of their regular transactions and form new queue, Q'.

Composing C_i from Q': A user u_i in Q' is added to C_i by the user in Q' from C_i if the new cluster C_i has an *AMR* value exceeding the given threshold. In the mean time, Removing the new user from Q'. Repeating the step until C_i has an AMR value lower than the given threshold.

Repeating the Last Step Until Q' is emptyset : If Q' is empty then HBC^2A has found all qualified clusters in C_1, ..., and C_i; Otherwise, HBC^2A has to repeat the last step.

The algorithm returns all the distributions that satisfy the requirements with the least number of application servers and let system administrators to decide which distribution they prefer.

4 An *AMR* 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 3.

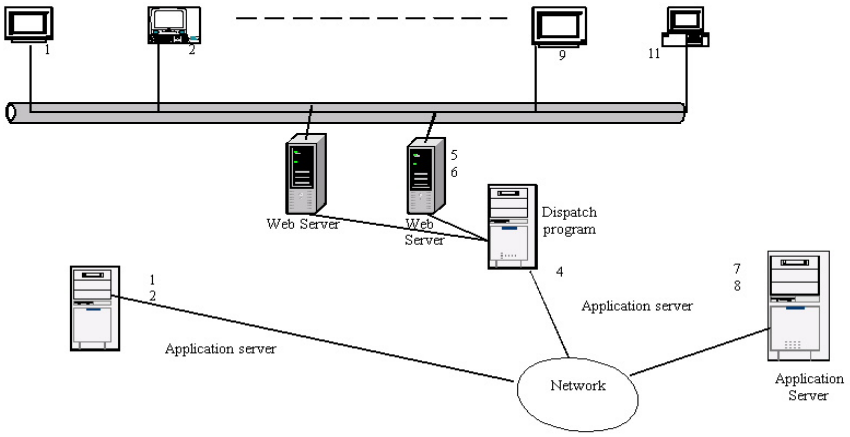

Fig. 3. Users are Distributed through a Dispatching Program

The distributions suggested by HBC^2A bases on frequent patterns in user profiles. For new and infrequent users, HBC^2A 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.

The running example is shown in figure 3. An *AMR* Based Hybrid dispatching algorithm distributes users while keeping the *AMR* of each server as high as possible. In the dispatching procedure, users are distributed to a server according to 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 transaction.
- Compute the potential new *AMR* in each server with the addition of the user. Assign the user to the server with the highest *AMR* , and update the *AMR* in the corresponding server.

The distribution in the running example has *AMR* s of 3/5, 3/4, and 1 in the three servers. If a new user with user-id 11 wishes to log on the system and submits an A as the first transaction then the user has a predicted regular transaction set of ABE, according to Table 3. The *AMR* after adding ABE to the three servers would be 3/5, 3/6, and 3/6, respectively. Because the first server has the highest AMR value, the new user is distributed to the first server, and the distribution becomes {1, 2, 3, 11}, {4, 5, 6}, and {7, 8, 9}.

5 Simulation

Several experiments are conducted on real data collected from a mid size machinery company based in Taichung, 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 are used to evaluate the quality of the suggested distributions.

In the experiment, 1,853,689 access logs are collected which include 56 users have regular patterns. The average number of transactions in user profiles is 7.7. The quality of suggested distributions are measured by Application Hit Ratios and Entropy. The Application Hit Ratio of a server is defined as the number of transaction accesses hits 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 in the distribution. 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. Since AR and AMR thresholds are typically smaller than 1, some frequent transactions are not stored in the memory. In the experiment, we assume that servers automatically store the applications that are accessed the most in the training data in the memory. Infrequent users appearing in the testing data are assigned to servers according to the hybrid distribution algorithm.

The Experiment of *HBC²A* and Round-Robin are 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 improved algorithm, distributions can be suggested within one minites.

5.1 Experimental Results of *HBC²A*

Seven distributions are suggested against the collected data. These distributions have *AMR* threshold set at 0.8 with AC ranging from 21 to 28, respectively. These simulations all have profile support threshold at 0.1 and user support threshold set at 0.3.

The number of servers needed for each distribution is shown in Figure 4. With *AMR* =0.8, *HBC²A* suggests a distribution with five servers when AC=21, four servers when AC = 22, three servers when AC = 23 , two servers when AC=24, 25, 26 and 27, and 1 server when AC = 28.

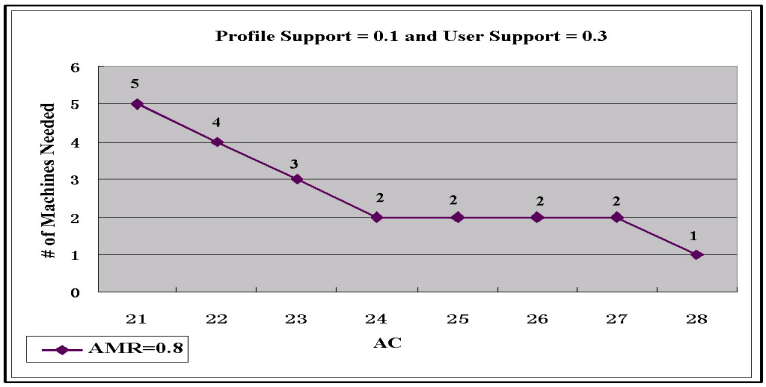

Fig. 4. Distribution Experiments with *HBC²A*

The quality of each distribution is evaluated in Figure 5 and Figure 6. The Application Hit Ratio of the distributions with AMR =0.8 is 0.91714 when AC = 21, 0.94924 when AC = 22, 0.95687 when AC = 23, 0.95674 when AC = 24 , 0.96696 when AC = 25, 0.96565 when AC = 26, 0.96474 when AC = 27 and 0.9559 when AC = 28. The Entropy of the distributions with AMR =0.8 is 6.6595 when AC = 21, 8.3807 when AC = 22, 9.542 when AC = 23, 10.0652 when AC = 24 , 10.0263 when AC = 25, 10.435 when AC = 26, 8.7664 when AC = 27 and 11.755 when AC = 28.

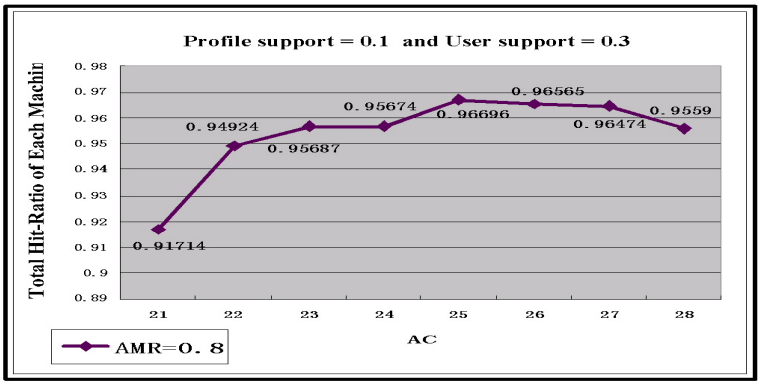

Fig. 5. The Application Hit Ratios of Distributions with HBC^2A

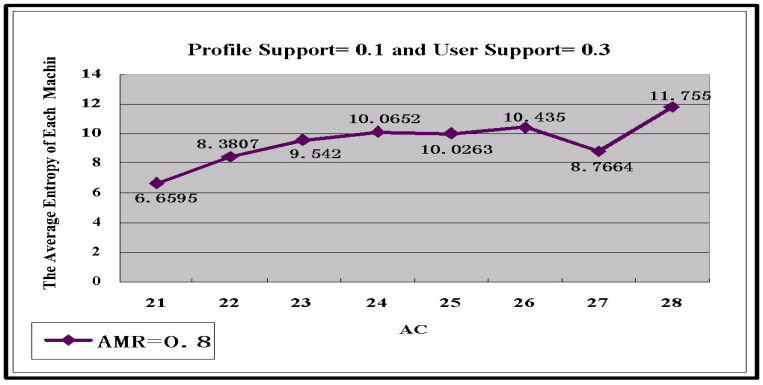

Fig. 6. The Entropy of Distributions with HBC^2A

From data shown in Figure 4, Figure 5, and Figure 6, we find that the hit-ratios of the distributions ranging from 0.91714 to 0.96696. From figure 5, we find that the more AC , the more average the higher hit-ratios machines have, because each machine can hold more transactions. The more number transactions manchine can hold, The more chances the high-ratios machine have. From figure 6, we find that the more AC, the more average entropy machines have. We conclude that the company should use one server to hold all users if hardware capacity is large enough. The second to the best distributions have Application Hit Ratios of 0.96696 which occurs when AMR =0.8, AC = 25(two machined need). Since the former settings requires fewer memory resource, system administrators are advised to adapt the former distribution.

5.2 Comparision of HBC^2A and Round-Robin User Distribution

HBC^2A considers the constraints and tries to find groups of users whose combinations of accessed transactions do not cause too many page faults if they are clustered into one application. Round-Robin distributes user to one of the several application servers in a server group by a rotated order. The approach ensure users are fairly distributed in a server group.

For the purpose of comparision, We set the same memory constraints to HBC^2A and Round-Robin. In terms of users and transactions allocations, HBC^2A can get better result than Round-Robin since given the same number of machines, the transaction distributions in HBC has lower entropy than Round-Robin, the hit-ratio of user distributed in each machine by HBC^2A get better result than Round-Robin. (Please refer figure 7, figure 8).

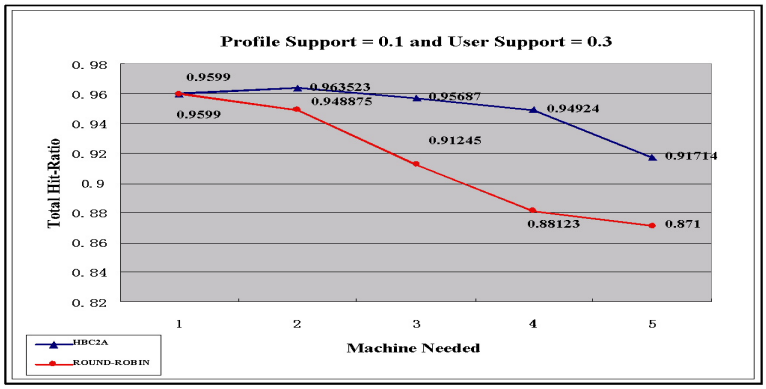

Fig. 7. Comparision of HBC^2A and Robin in Hit Ratios of the Experiment

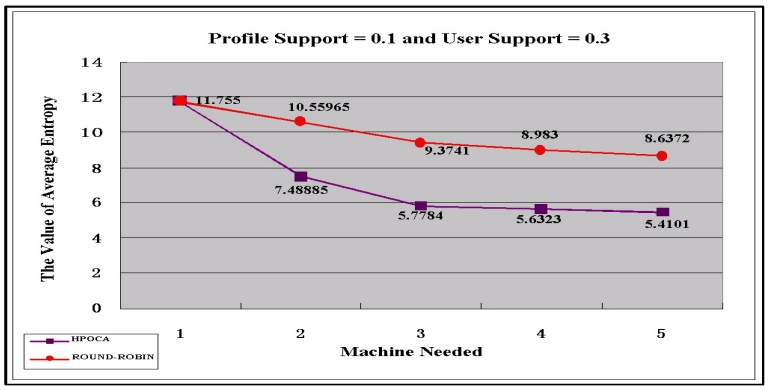

Fig. 8. Comparision of HBC^2A and Robin in Entropy of the Experiment

In summary, if the memory of each machine is seriously limited , HBC^2A should be used to distribute users, because the result generated by them is guaranteed to satisfy the memory consumption criteria.

6 Related Work

With the Internet rush, many researches have been devoted to distribute user requests in Distributed Web Server Architecture, in order to improve the performance of web servers. Depending on the locations where

request distributions happen, these researches are classified in client-based, DNS (Domain Name Server)-based, dispatcher-based, and server-based, as in [6, 5, 18, 8, 20]. Since current `Http protocol` is stateless, each request is routed independently to a web server[5, 3, 16, 17, 19]. All of the above researches assume that requests can be independently routed to different servers, where as in the application servers of ESs, requests from the same users have to be routed to the same server.

Clustering literatures are classified into two models: partitioning clustering and hierarchical clustering [15, 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 reached. Hierarchical clustering can be further divided into agglomerative and divisive clustering. As the name suggested, agglomerative clustering 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.

Most clustering algorithms employ Euclidean distances to compute similarity. The shorter the distances the more 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 $\langle 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 \rangle$. 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 $\langle 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0 \rangle$. Since {A} and {I} have a distance of two bits, and {A} and {A, B, E, F} have a distance of three bits, 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 categorial data.

Many set oriented algorithms use Jaccard coefficient [15] 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 has two drawbacks for our application. The first is that it cannot describe the number of elements in each cluster, which are important to calculate the buffer efficiency. The second is that Jaccard coefficient is not accurate in computing the similarity between transactions sets. For example, the Jaccard coefficient of {A,B,C},{A} and {A,B,C},{B,C,D} are 1/3 and 2/4, respectively. However, in *HBC*^{2}*A* , the distance of the former pair is 0, since {A, B, C} include {A}. 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. For instance, if a profile includes transaction sets {A}, {A, B, C},{A, C, D},{B, C, D},{B, C, E} and the threshold of a qualified common neighbor(link) is set at 1/3 of Jaccard coefficient. The ROCK coefficient of {A,B,C},{A} and {A,B,C},{B,C,D} are 1 (due to the common neighbor {A, C, D}) and 2 (due to the common neighbor {A, C, D} and {B, C, E}), respectively. From view of ROCK, the similarity of the former pair is lower than later pair. On the other hand, the distance of the former pair is 0 in *HBC*^{2}*A* , which is more close to our intuition in the application of user distribution, since {A} is a subset of {A, B, C}. Many set oriented algorithms use Jaccard coefficient and ROCK. However, Jaccard coefficient and ROCK along cannot describe the number of elements in each cluster, which are

important to calculate the buffer efficiency. Hence, common categorial clustering technology is not suitable for clustering users in the application.

7 Conclusion

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

Since ESs tend to consume considerable amount of hardware memory, application servers can easily run out all memory available, which induce to hardware limitations. When this happens, a common procedure adopted in boosting performance is adding application servers to ESs. With multiple application servers in the scene, distributing users with similar application requirements to the same application servers increases buffer utilization and lead time to next hardware upgrades.

The procedure of HBC^2A roots its development on AMR , which is a similarity measure of user transactions grouped in the same cluster. A cluster with high AMR means users in the cluster share similar applications under a given buffer limitation. AMR has the property of Anti-Monotonicity, which states that AMR of a cluster decreases with the addition of each new transaction set. With the property, HBC^2A can prune hopeless search branches and stop the iterations when an empty cluster set is found. Distributions are combinations of clusters which cover all users with regular transactions and each user is included in only one cluster. The distributions composed of fewest number of clusters are returned as suggestions.

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 A AMRs. 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 associated with the user. The associated transactions are then used to decide the target server for the new user. The user goes to the server with the highest AMR after accepting the user.

As future work, HBC^2A is among a series of study in distributing users with historical user profiles, and are by no means the last two. Several issues require further studies, such as modelling user profiles with sequences, dynamically updating user patterns, incorporating CPU and systems loads into dispatching and distribution algorithms.

Acknowledgements

This study is supported by National Science Council, Taiwan, Republic of China, through the Project No.NSC94-2416-H-029-010-. 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. 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.
5. H. Bryhni, E. Klovning, and O. Kure. A comparison of load balancing techniques for scalable web servers. *IEEE Network*, 14:58–64, 2000.
6. V. Cardellini, M. Colajanni, and P.S. Yu. Dynamic load balancing on web-server systems. *IEEE Internet Computing*, 3:28–39, 1999.
7. Yen-Liang Chen, Ping-Yu Hsu, and Chun-Ching Ling. Mining quantitative assocation rules in bag databases. *Journal of Information Management*, 7:215–229, 2001.
8. Gianfranco Ciardo, Alma Riska, and Evgenia Smirni. Equiload:a load balancing policy for cluster web servers. *Performance Evaluation*, 46:101–124, 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 Clustersing. Morgan Kaufmann Publisher, 2001.
13. J.A. Hernándes. *The SAP R/3 Handbook*, chapter Distributing R/3 Systems. McGraw-Hill, 2 edition, 2000.
14. J. Pei J. Han and Y. Yin. Mining frequent patterns without candidate generation. In *Proceedings of ACM-SIGMOD International Conference on Management of Data*, pages 1–12, 2000.
15. A.K. Jain and R.C. Dubes. *Algorithms for Clustering Data*. Prentice Hall, 1988.
16. 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.
17. S. Nadimpalli and S. Majumdar. Techniques for achieving high performance web servers. In *Proceedings of International Conference on Parallel Processing*, pages 233–241, 2000.
18. 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.
19. Victor Safronov and Manish Parashar. Optimizing web servers using page rank prefetching for clustered accesses. *Information Sciences*, 150:165–176, 2003.
20. Zhiguang Shan, Chuang Lin, and Dan Marineslu. Modeling and performance analysis of qos-aware load balancing of web-server cluster. *Computer Networks*, 40(2):235–244, 2002.

出席國際會議報告書

報告人：丁冰和

私立東海大學餐旅管理系

2006.08.20

一、 參加會議經過

今年的The Seventh International Conference on Web-Age Information Management在香港理工學院舉辦，這個會議涵蓋的內容相當廣泛，從Advanced application of databases、Biological and genome information systems、Data mining and knowledge discovery、Web service and information management、Mobile data management、Workflow and E-services，包括25種研究方向，世界各地的學者在此齊聚一堂，交換研究心得，會期由6/17/2006到6/19/2006共3天，除了Lecture外，還有Workshop、tutorial和Keynote Speech，第一天主要是Workshop的舉行，第二天及第三天才有論文的發表，但也都有安排keynote speaker演講，本次研討會所接受的論文將刊登到Lecture Notes in Computer Science(LNCS)，具有SCI INDEX，所以論文品質是有一定水準的，參與程度相當踴躍。 在會場觀摩其他學者的論文，海報發表以觀察海報格式、和作者交流為要點，口頭報告則以發表者的表達技巧、研究方向、和台下聽眾問答情況為觀察重點。 面對網際網路的快速發展，本研會從設計面，應用面及管理面來探討以web-based的資訊系統(從理論到實務），本人有一篇論文和本研會相關的研究，關於Parallel and distributed database systems，題目為：Development of User Distributions in Enterprise Systems with limited Buffer Sizes in Application Servers，文章將放於後面，供委員們參考。

二、 與會心得

出席國際研討會最能增長自己的見聞，看到國外的學者對他們的研究的敬業及報告時專注並專業的的眼神，令人敬佩，演講者所展現的powerpoint做得很漂亮也很專業，配合著流利的思維與口才，一方面感受到自己語文能力的不足，不免激勵自己加強語文能力的心，有時會有聽不懂與停頓的情形，有時和別人溝通自己的想法，有時會有表答不順暢或說不清楚的情形發生。參加研討和來自全世界各地的朋友交換名片是一個很重要的事，藉著這樣的機會大家彼此交換資訊與心得，以利研究的澄清及後續研究的共同努力，是一個很好的機會,正式的論文討論及休息期間的餐會都是一個很好把握的場合。

因為是第二次參加國際會議，還在累積經驗，往往無法一眼看透其他人研究的重點所在，當keynote speaker演講時，人數還算多，但分開場地發表論文，尤其主辦單位還分不同主題不同場地，再加錄取的論文不到50篇，使得發表的現場顯得聽的人並不會很多，有些人或許因為太緊張而導致在台上吞吞吐吐，有些人則能夠非常清楚明白的將自己的研究向聽眾分享。對我來說，用

英文和外國學者交換心得是很新鮮的體驗，因為世界各地口音各各不同，有時候很難聽懂，有時候自己也會詞不達意，回國後必須要好好的加強英文，希望很快能夠再度出國見世面長見識。

另外，我也看到香港這城市都市化及國際化的情形，令人欣賞，地鐵捷運四通八達非常方便，有英語國語客語等廣播及指示標誌非常完全，此地雖為海島地型盆地，高低起浮不平，但到處都有電扶梯輔助，非常方便。此外，也看到金融體系的發達及世界貿易展的人潮，另外抽空參觀科學館及歷史博物館，做得非常詳細與用心，令人印象深刻

三、 建議

主辦單位在會場安排上並不完善，場地分散在相當多的學院及房間，不能輕易依指示找到位置，我繞很多地點才找到研討會位止，且進入校園的每一個 building皆有警衛看守，皆要出示證件登記，相當不便，休息期間想要進入圖書館參觀，竟然被拒絕，顯然不受到尊重，殊不知全校共同體的觀念，學術單位辦研討會，行政單位應配合以利形象的建立。

四、 攜回資料名稱及內容

1. 6個重要研討會相關訊息及會議相關資料
a. APWEB/WAIM 2007
b. The 2006 IEEE International Conference on Data Mining
c. The fifth International Conference on Web-based Learning
d. Fourth International Conference on Cooperative Internet Computing
e. 2nd International Conference on Trends in Enterprise Application Architecture
f. 26th ACM SIGMOD International Conference on Management of Data

Development of Users Distribution in Enterprise Systems with limited Buffer Size in Application Servers

Ping-Ho Ting[1], Kuan-Ching Li[2], and Chun Chung Wei[3]

[1] Dept. of Hospitality Management, TungHai University
Taichung 40704, Taiwan
ding@thu.edu.tw
[2] Dept. of Computer Science and Information Management, Providence University
Shalu, Taichung 43301, Taiwan
kuancli@pu.edu.tw
[3] Dept. of Information Management,ChungChou Institute of Technology
Yuanlin, Chunghwa 51003, Taiwan
ccwei@ms15.url.com.tw

Abstract. As enterprises worldwide race to improve their real-time management turnaround, which is essential requirement to improvements in productivity and service deployments, and therefore, large amount of resources have been invested into Enterprise Systems (ESs). All modern and robust ESs adopt a n-tier client-server architecture, which includes several application servers to hold users and applications. Currently, most web systems are stateless, which means that each request is routed independently to a different server at each time. However, for ESs, each request from same user is routed to the same application server.

Distributions in application and web servers are different in granularity. In the former scenario, a user represented by a set of transactions is the atomic element, while in the latter scenario, single request is the atomic element and different requests issued by the same user can be directed to different web servers. Until present time, few researches have been devoted in the user distribution to application servers in n-tier architecture.

In this paper, it is proposed a Heuristic Buffer Constraint Clustering Algorithm, namely HBC^2A , which is a Greedy-based strategy algorithm. The algorithm give suggestions of user distributions, the number of servers needed, and the similarity of user requests in each server. In addition, this algorithm is applied on a set of real data which is derived from the access log of an Enterprise System, in order to evaluate the quality of suggested distributions.

1 Introduction

For ESs that process daily business transactions, users typically have low tolerance on system performance. If a system responds to data entries or queries too slowly, users lose patience and complain loudly. Yet, the number of ES users keeps growing in most companies as the number of business processes incorporated into ESs increases. Therefore, keeping response time under control is a vital issue for most system administrators. To boost performances, activating more than one application servers become common practices in the industry.

When an ES has multiple application servers, distributing users to similar applications and application servers plays an important role in tuning overall system performance, as pointed out by documents of major ES systems [1, 13]. Throughout our text, an application in an ES corresponds to an atomic and unbreakable transaction, transactions and applications are used interchangeably.

In current practices, ESs do not automatically switch users to other application servers, due to the resources involved in the transmissions. As a user logs onto an application server, all related data such as authorizations, preferences, and created data are collected in the server's virtual memory, in order to create time-sharing working environment and reduce user effort in keying data. Besides, all applications executed by users are compiled and stored in memory. Data accessed by applications are also cached in memory to

improve the efficiency of systems. In many cases, the amount of data cached are huge, and as consequence, transferring a user to a different application server may trigger a transmission of huge amount of data. Thus, users are not switched automatically among servers in current practice.

In ESs, the dispatching mechanism needs to consider two criteria to gain reasonable performance: the number of users log on in each application server and the collection of applications executed in servers. Therefore, as each user consume hardware resources and the n-tier architecture has more than one application server, user distribution becomes one of the important issues in tuning ES performance [2]. In ESs, each application is evoked by a user who logs on an application server, and stays connected to the server for an entire working session, which can last for several hours and includes the execution of a set of applications. Therefore, admitting a user into an application server is equivalent to admitting a set of transactions into an application server, which marks a sharp difference between the distribution of application severs and traditional web servers.

In traditional web servers, requests are examined individually and those issued by the same user can be routed to different web servers. Commercial products, such as SAP R/3, equipped with a simple dispatching algorithm, considers only number of users and server response time. The task of grouping users is left to system administrators [1, 13]. In addition to the rough guideline of grouping financial users into one server and logistic users into another, system administrators need specific suggestions, such as explicit user distributions, the number of servers needed and the similarity of user requests in each server. To address the needs, this research paper proposes algorithm to suggest distribution based on user profiles. The distribution algorithm can the least number of servers needed that satisfy all the constraints of a system.

The scheme of the proposed research is shown in figure 1. The procedure is started with the collection of user profiles from an enterprise system. The profile is consisted of a set of transactions accessed by users in a given period of time. The transactions that are accessed frequently are labelled as regular transactions. The frequent accesses are compared against profile support threshold and user support threshold. The purpose of profile support is to screen transactions that are seldom used by all users and user support threshold is to find transactions which are accessed frequently by each 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 predicted the behavior of new and not frequent users, who do not have enough records in the user profile. In the distribution, regular transactions are used to cluster users with the novel algorithm prosed in this research paper, namely HBC^2A .

To explain the algorithms and related procedures, the rest of the paper is organized into the following sections. Applications are grouped into large itemsets with traditional Apriori algorithm[7, 11] to find frequent patterns. The process is explained in section 2. A group of users forms a cluster if the union of the users' transaction sets has an Application Match Ratio(AMR) exceeds a given threshold. AMR is a similarity measure of user patterns grouped in the same set. The definition of AMR and related properties are proved in section 3. An example of AMR based hybrid distributing approach is shown in section 4 Simulations with real data and comparisions with Round-Robin users distribution are shown in section 5. A review of distributed web server architectures and clustering of categorical sets is shown in section 6. Conclusion and possible extension of HBC^2A are discussed in section 7.

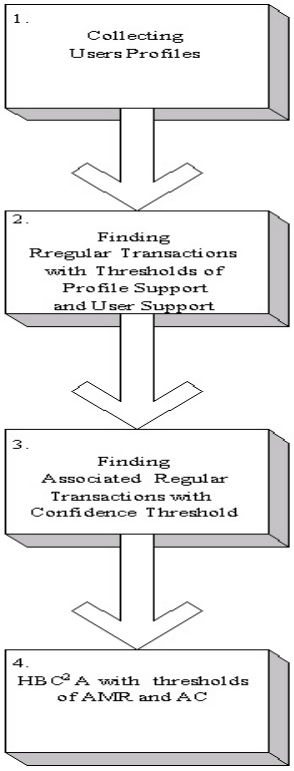

Fig. 1. Research scheme.

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 sessions. For the purpose of the paper, these data are transformed into user profiles. A user profile is a set of $\langle user - id, transaction - set \rangle$, 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 1, 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.

Table 1. User Profiles

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}

As careful readers may have found that the transactions accessed by user 10 in the profile shown in Table 1 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. 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*

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

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

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 [4, 14]. 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 2 shows the stages in computing regular transactions.

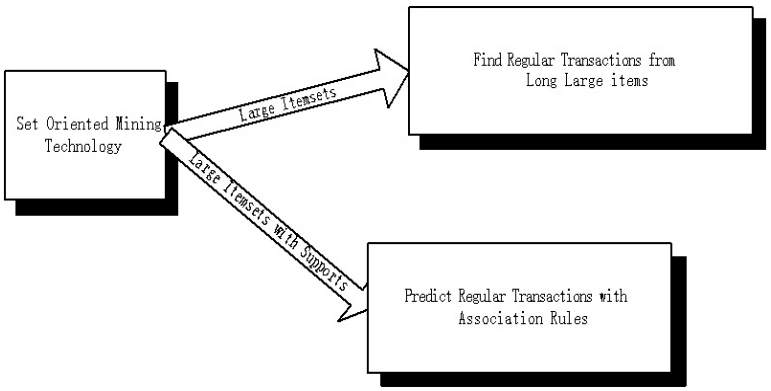

Fig. 2. The Stages of Computing Regular Transactions

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 the running example is shown in Table 2.

Table 2. Regular Transactions

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	∅

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 logging on, the dispatching program can check if the transaction has high association with any large itemsets. If so, the union of the large itemsets dentoe the user's 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*

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

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 3.

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 Clustering and Distributing by *HBC²A*

Systems with multiple servers gain performance speed at the cost of keeping duplicated programs and data in more than one servers. In sophisticated application servers with hundreds or thousands of users on-line all the time, the memory needed are considerable [2]. Therefore, users share similar transactions are grouped

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

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%

into one cluster, which is then assigned to an application server. This section proposes HBC^2A to cluster users and a straightforward algorithm to distribute clusters.

Definition 3 *A cluster is a set of users that share common applications in an Enterprise system.*

The quality of a cluster is measured by AMR , Application Match Ratio. The AMR of a cluster is defined as the ratio of AC versus the applications in the cluster, where AC denotes the number of applications that can be hosted in an application server without causing buffer swap. AMR is smaller than one when users in the cluster have more regular transactions than the buffers can hold. In this case, buffer swap occurs and the smaller the AMR is, the more the buffer swap will occur.

Definition 4 *The AC of an enterprise system is an integer number. The number denotes the number of applications that can resides in application servers of the enterprise systems without causing buffer swap.*

The regular transactions in a cluster are defined as the union of regular transactions of users grouped in the cluster.

Definition 5 *The number of regular transactions in a cluster, c, is defined as*

$$\|c\| = |\cup_{u \in c}\, u.regular\ transactions|$$

The AMR of a cluster, c, is defined as the ratio of AC to $\|c\|$. $AMR\ (c) = \frac{AC}{\|c\|}$.

Lemma 1 *The AMR of each cluster has a value between 0 and AC .*

Proof

AMR 's are positive and therefore are always greater than to 0.

Given a cluster, c

$$\begin{aligned} AMR\ (c) &= \frac{AC}{\|c\|} \\ &\leq \frac{AC}{1} \\ &\leq AC \end{aligned}$$

AMR of a cluster, therefore has values between 0 and AC .

□

Hence, system administrators can assign an AMR threshold between 0 and AC . By setting the threshold is between 0 and AC , the system administrators can tune the tolerance degree of buffer overflow.

Theorem 1 Anti-Monotonicity of AMR *AMR of a cluster decreases with the addition of any user with non-empty regular transaction set to the cluster.*

Proof

If a cluster, c, has the AMR of $\frac{AC}{p}$ *where p is the number of different transactions in the cluster. If a user with q new transactions is added to the cluster then the new AMR is* $\frac{AC}{p+q}$.

$$\begin{aligned}\frac{AC}{p} - \frac{AC}{p+q} &= \frac{AC * (p+q) - AC * p}{p * (p+q)} \\ &= \frac{AC * q}{p * (p+q)} \\ &\geq 0\end{aligned}$$

The case of $\frac{AC*q}{p*(p+q)} = 0$ *occurs when q=0, which means the regular transaction set of the new user does not contain any new transactions.*

□

Therefore, AMR has the property of Anti-Monotonicity, which means that adding a user to a cluster can only reduce the AMR of the cluster, unless the new transaction set does not contain any new transactions. The property can be used to prune hapless candidate clusters that have AMR under a threshold in the cluster forming algorithm, HBC^2A . In this paper, system administrators are requested to supply an AMR threshold. Candidate clusters with AMR smaller than the threshold are discarded.

Theorem 2 *The threshold of AMR must be smaller than or equal to* $\frac{AC}{|t_{max}|}$ *, where* t_{max} *is the largest regular transaction set in the user profile , to have all users grouped into at least one cluster.*

Proof
Any cluster c containing users with t_{max} *has AMR (c)* $\leq \frac{AC}{|t_{max}|}$. *If the threshold is larger than* $\frac{AC}{|t_{max}|}$, *then the users can not be included in any cluster.*

□

Definition 6

– *A qualified cluster is a cluster whose AMR exceeds a given threshold.*

- *A set of clusters is comprehensive under a user profile, U, if the union of the clusters includes all users with regular transactions in U.*
- *A set of clusters is disjoined if the intersections of any two clusters are empty.*
- *A set of qualified clusters is a distribution under a user profile, U, if they are comprehensive under U and disjoined.*

In the running example, if *AC* is set at 3, and *AMR* threshold at 0.5 , then the cluster of {1,2,3} ,{4,5,6} and {7,8,9} have *AMR* of 0.6, 0.75 and 1, respectively. The set composed by the three clusters is comprehensive, disjoined and forms a valid distribution. The running example is shown in Table 4.

Table 4. A set of qualified clusters when AC=3 and AMR=0.5

Qualified cluster	Users	Regular Transactions	AMR
Cluster 1	1,2,3	A,B,E,F,H	0.6
Cluster 2	4,5,6	B,I,J,K	0.75
Cluster 3	7,8,9	P,Q,R	1

We propose a Heuristic BC^2A , namely HBC^2A , HBC^2A returns distributions that satisfy constraints with the fewest number of clusters, and the rules associating single transactions to predicted regular transactions. The constraints include *AC* , an *AMR* threshold, profile support threshold, user support threshold, and rule confidence threshold. The recommendations guarantee that when all frequent users logging on the system and accessing all regular transactions, each server still has an *AMR* above the given *AMR* Threshold. Information included in the recommendations are number of servers, clusters of users, and *AMR*s of clusters.

The HBC^2A includes three steps in computing the recommendations - computing the set of qualified clusters and selecting clusters to form distribution. The main steps are listed as following:

Initialization: for each user with regular transactions, and these users form qure, Q. Sort Q on users by the number of their regular transactions and form new queue, Q'.

Composing C_i from Q': A user u_i in Q' is added to C_i by the user in Q' from C_i if the new cluster C_i has an *AMR* value exceeding the given threshold. In the mean time, Removing the new user from Q'. Repeating the step until C_i has an AMR value lower than the given threshold.

Repeating the Last Step Until Q' is emptyset : If Q' is empty then HBC^2A has found all qualified clusters in C_1, ..., and C_i; Otherwise, HBC^2A has to repeat the last step.

The algorithm returns all the distributions that satisfy the requirements with the least number of application servers and let system administrators to decide which distribution they prefer.

4 An *AMR* 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 3.

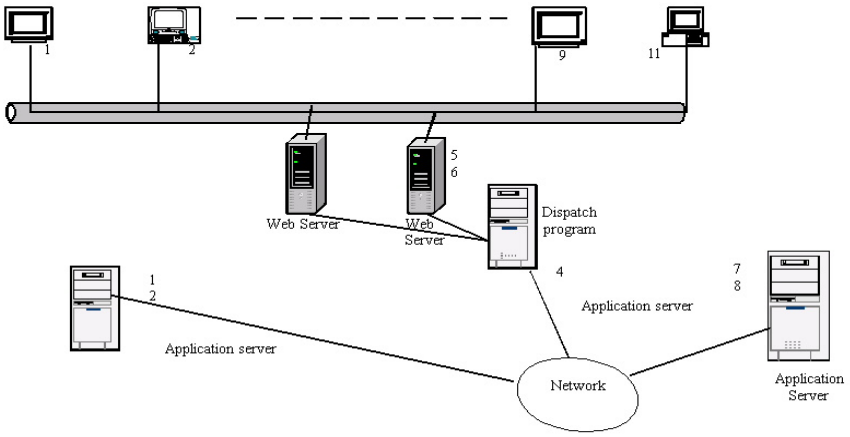

Fig. 3. Users are Distributed through a Dispatching Program

The distributions suggested by HBC^2A bases on frequent patterns in user profiles. For new and infrequent users, HBC^2A 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.

The running example is shown in figure 3. An *AMR* Based Hybrid dispatching algorithm distributes users while keeping the *AMR* of each server as high as possible. In the dispatching procedure, users are distributed to a server according to 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 transaction.
- Compute the potential new *AMR* in each server with the addition of the user. Assign the user to the server with the highest *AMR* , and update the *AMR* in the corresponding server.

The distribution in the running example has *AMR* s of 3/5, 3/4, and 1 in the three servers. If a new user with user-id 11 wishes to log on the system and submits an A as the first transaction then the user has a predicted regular transaction set of ABE, according to Table 3. The *AMR* after adding ABE to the three servers would be 3/5, 3/6, and 3/6, respectively. Because the first server has the highest AMR value, the new user is distributed to the first server, and the distribution becomes {1, 2, 3, 11}, {4, 5, 6}, and {7, 8, 9}.

5 Simulation

Several experiments are conducted on real data collected from a mid size machinery company based in Taichung, 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 are used to evaluate the quality of the suggested distributions.

In the experiment, 1,853,689 access logs are collected which include 56 users have regular patterns. The average number of transactions in user profiles is 7.7. The quality of suggested distributions are measured by Application Hit Ratios and Entropy. The Application Hit Ratio of a server is defined as the number of transaction accesses hits 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 in the distribution. 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. Since AR and AMR thresholds are typically smaller than 1, some frequent transactions are not stored in the memory. In the experiment, we assume that servers automatically store the applications that are accessed the most in the training data in the memory. Infrequent users appearing in the testing data are assigned to servers according to the hybrid distribution algorithm.

The Experiment of *HBC²A* and Round-Robin are 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 improved algorithm, distributions can be suggested within one minites.

5.1 Experimental Results of *HBC²A*

Seven distributions are suggested against the collected data. These distributions have *AMR* threshold set at 0.8 with AC ranging from 21 to 28, respectively. These simulations all have profile support threshold at 0.1 and user support threshold set at 0.3.

The number of servers needed for each distribution is shown in Figure 4. With *AMR* =0.8, *HBC²A* suggests a distribution with five servers when AC=21, four servers when AC = 22, three servers when AC = 23 , two servers when AC=24, 25, 26 and 27, and 1 server when AC = 28.

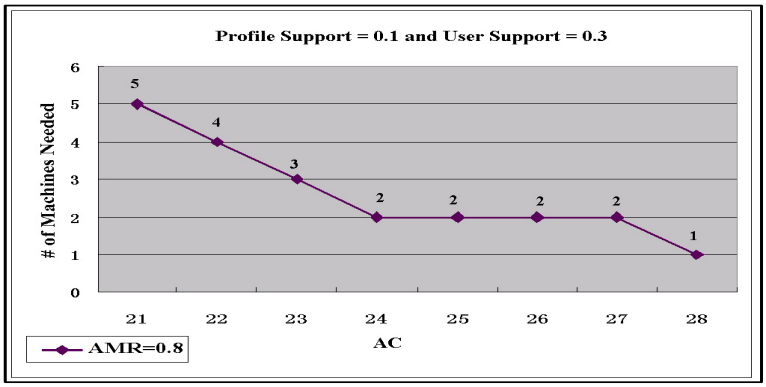

Fig. 4. Distribution Experiments with *HBC²A*

The quality of each distribution is evaluated in Figure 5 and Figure 6. The Application Hit Ratio of the distributions with AMR =0.8 is 0.91714 when AC = 21, 0.94924 when AC = 22, 0.95687 when AC = 23, 0.95674 when AC = 24 , 0.96696 when AC = 25, 0.96565 when AC = 26, 0.96474 when AC = 27 and 0.9559 when AC = 28. The Entropy of the distributions with AMR =0.8 is 6.6595 when AC = 21, 8.3807 when AC = 22, 9.542 when AC = 23, 10.0652 when AC = 24 , 10.0263 when AC = 25, 10.435 when AC = 26, 8.7664 when AC = 27 and 11.755 when AC = 28.

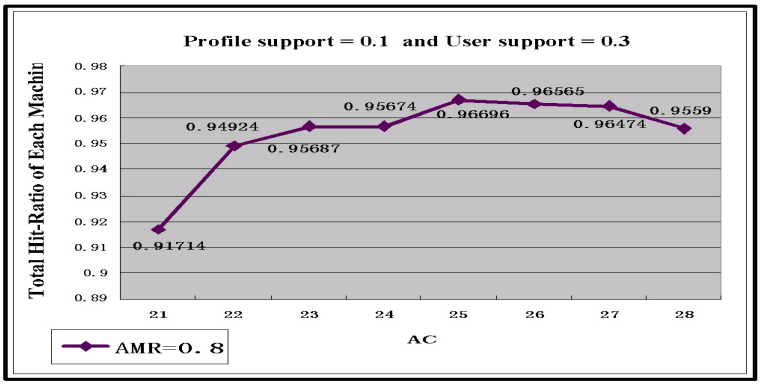

Fig. 5. The Application Hit Ratios of Distributions with HBC^2A

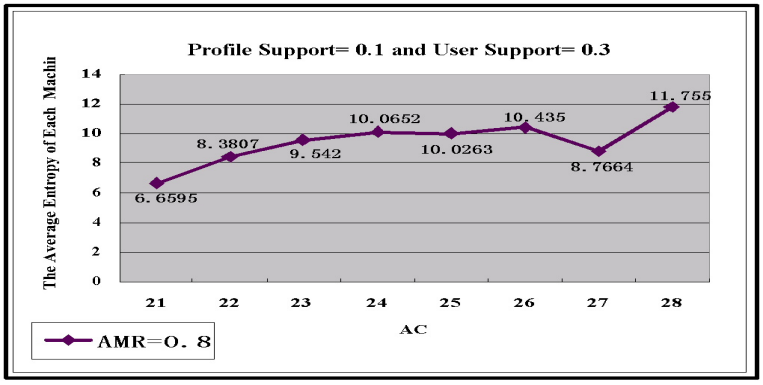

Fig. 6. The Entropy of Distributions with HBC^2A

From data shown in Figure 4, Figure 5, and Figure 6, we find that the hit-ratios of the distributions ranging from 0.91714 to 0.96696. From figure 5, we find that the more AC , the more average the higher hit-ratios machines have, because each machine can hold more transactions. The more number transactions manchine can hold, The more chances the high-ratios machine have. From figure 6, we find that the more AC, the more average entropy machines have. We conclude that the company should use one server to hold all users if hardware capacity is large enough. The second to the best distributions have Application Hit Ratios of 0.96696 which occurs when AMR =0.8, AC = 25(two machined need). Since the former settings requires fewer memory resource, system administrators are advised to adapt the former distribution.

5.2 Comparision of HBC^2A and Round-Robin User Distribution

HBC^2A considers the constraints and tries to find groups of users whose combinations of accessed transactions do not cause too many page faults if they are clustered into one application. Round-Robin distributes user to one of the several application servers in a server group by a rotated order. The approach ensure users are fairly distributed in a server group.

For the purpose of comparision, We set the same memory constraints to HBC^2A and Round-Robin. In terms of users and transactions allocations, HBC^2A can get better result than Round-Robin since given the same number of machines, the transaction distributions in HBC has lower entropy than Round-Robin, the hit-ratio of user distributed in each machine by HBC^2A get better result than Round-Robin. (Please refer figure 7, figure 8).

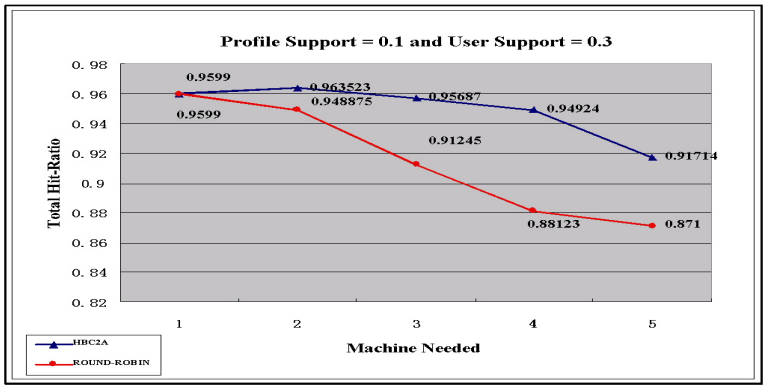

Fig. 7. Comparision of HBC^2A and Robin in Hit Ratios of the Experiment

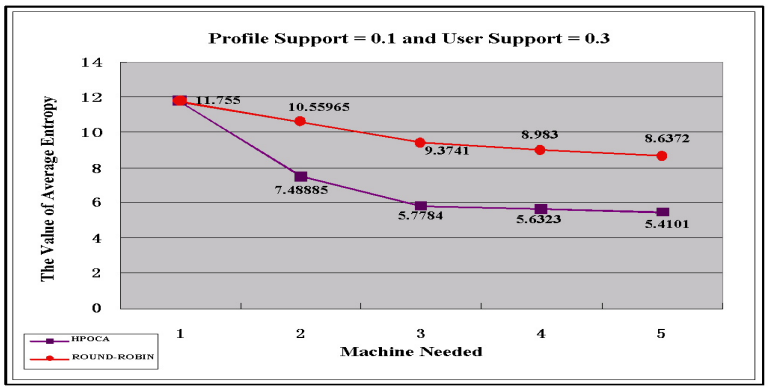

Fig. 8. Comparision of HBC^2A and Robin in Entropy of the Experiment

In summary, if the memory of each machine is seriously limited , HBC^2A should be used to distribute users, because the result generated by them is guaranteed to satisfy the memory consumption criteria.

6 Related Work

With the Internet rush, many researches have been devoted to distribute user requests in Distributed Web Server Architecture, in order to improve the performance of web servers. Depending on the locations where

request distributions happen, these researches are classified in client-based, DNS (Domain Name Server)-based, dispatcher-based, and server-based, as in [6, 5, 18, 8, 20]. Since current `Http protocol` is stateless, each request is routed independently to a web server[5, 3, 16, 17, 19]. All of the above researches assume that requests can be independently routed to different servers, where as in the application servers of ESs, requests from the same users have to be routed to the same server.

Clustering literatures are classified into two models: partitioning clustering and hierarchical clustering [15, 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 reached. Hierarchical clustering can be further divided into agglomerative and divisive clustering. As the name suggested, agglomerative clustering 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.

Most clustering algorithms employ Euclidean distances to compute similarity. The shorter the distances the more 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 $\langle 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 \rangle$. 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 $\langle 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0 \rangle$. Since {A} and {I} have a distance of two bits, and {A} and {A, B, E, F} have a distance of three bits, 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 categorial data.

Many set oriented algorithms use Jaccard coefficient [15] 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 has two drawbacks for our application. The first is that it cannot describe the number of elements in each cluster, which are important to calculate the buffer efficiency. The second is that Jaccard coefficient is not accurate in computing the similarity between transactions sets. For example, the Jaccard coefficient of {A,B,C},{A} and {A,B,C},{B,C,D} are 1/3 and 2/4, respectively. However, in *HBC*^{2}*A* , the distance of the former pair is 0, since {A, B, C} include {A}. 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. For instance, if a profile includes transaction sets {A}, {A, B, C},{A, C, D},{B, C, D},{B, C, E} and the threshold of a qualified common neighbor(link) is set at 1/3 of Jaccard coefficient. The ROCK coefficient of {A,B,C},{A} and {A,B,C},{B,C,D} are 1 (due to the common neighbor {A, C, D}) and 2 (due to the common neighbor {A, C, D} and {B, C, E}), respectively. From view of ROCK, the similarity of the former pair is lower than later pair. On the other hand, the distance of the former pair is 0 in *HBC*^{2}*A* , which is more close to our intuition in the application of user distribution, since {A} is a subset of {A, B, C}. Many set oriented algorithms use Jaccard coefficient and ROCK. However, Jaccard coefficient and ROCK along cannot describe the number of elements in each cluster, which are

important to calculate the buffer efficiency. Hence, common categorial clustering technology is not suitable for clustering users in the application.

7 Conclusion

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

Since ESs tend to consume considerable amount of hardware memory, application servers can easily run out all memory available, which induce to hardware limitations. When this happens, a common procedure adopted in boosting performance is adding application servers to ESs. With multiple application servers in the scene, distributing users with similar application requirements to the same application servers increases buffer utilization and lead time to next hardware upgrades.

The procedure of HBC^2A roots its development on AMR , which is a similarity measure of user transactions grouped in the same cluster. A cluster with high AMR means users in the cluster share similar applications under a given buffer limitation. AMR has the property of Anti-Monotonicity, which states that AMR of a cluster decreases with the addition of each new transaction set. With the property, HBC^2A can prune hopeless search branches and stop the iterations when an empty cluster set is found. Distributions are combinations of clusters which cover all users with regular transactions and each user is included in only one cluster. The distributions composed of fewest number of clusters are returned as suggestions.

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 A AMRs. 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 associated with the user. The associated transactions are then used to decide the target server for the new user. The user goes to the server with the highest AMR after accepting the user.

As future work, HBC^2A is among a series of study in distributing users with historical user profiles, and are by no means the last two. Several issues require further studies, such as modelling user profiles with sequences, dynamically updating user patterns, incorporating CPU and systems loads into dispatching and distribution algorithms.

Acknowledgements

This study is supported by National Science Council, Taiwan, Republic of China, through the Project No.NSC94-2416-H-029-010-. 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. 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.
5. H. Bryhni, E. Klovning, and O. Kure. A comparison of load balancing techniques for scalable web servers. *IEEE Network*, 14:58–64, 2000.
6. V. Cardellini, M. Colajanni, and P.S. Yu. Dynamic load balancing on web-server systems. *IEEE Internet Computing*, 3:28–39, 1999.
7. Yen-Liang Chen, Ping-Yu Hsu, and Chun-Ching Ling. Mining quantitative assocation rules in bag databases. *Journal of Information Management*, 7:215–229, 2001.
8. Gianfranco Ciardo, Alma Riska, and Evgenia Smirni. Equiload:a load balancing policy for cluster web servers. *Performance Evaluation*, 46:101–124, 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 Clustersing. Morgan Kaufmann Publisher, 2001.
13. J.A. Hernándes. *The SAP R/3 Handbook*, chapter Distributing R/3 Systems. McGraw-Hill, 2 edition, 2000.
14. J. Pei J. Han and Y. Yin. Mining frequent patterns without candidate generation. In *Proceedings of ACM-SIGMOD International Conference on Management of Data*, pages 1–12, 2000.
15. A.K. Jain and R.C. Dubes. *Algorithms for Clustering Data*. Prentice Hall, 1988.
16. 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.
17. S. Nadimpalli and S. Majumdar. Techniques for achieving high performance web servers. In *Proceedings of International Conference on Parallel Processing*, pages 233–241, 2000.
18. 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.
19. Victor Safronov and Manish Parashar. Optimizing web servers using page rank prefetching for clustered accesses. *Information Sciences*, 150:165–176, 2003.
20. Zhiguang Shan, Chuang Lin, and Dan Marineslu. Modeling and performance analysis of qos-aware load balancing of web-server cluster. *Computer Networks*, 40(2):235–244, 2002.