Journal of Knowledge Information Technology and Systems

ISSN 1975-7700

http://www.kkits.or.kr

Selectivity Estimation Using Frequent Itemset Mining

Boyun Eom[1], Christopher Jermaine[2], Choonhwa Lee[*3]

[1]*Intelligent Convergence Technology Research Department, ETRI*
[2]*Department of Computer Science, Rice University*
[3]*Division of Computer Science and Engineering, Hanyang University*

ABSTRACT

In query processing, query optimization is an important function of a database management system since overall query execution time can be significantly affected by the quality of the plan chosen by the query optimizer. Under cost-based optimization, a query optimizer estimates the cost for every possible query plans based on the underlying data distribution in synopses of database relations. The most common synopses in commercial databases have been histograms. However, when there is correlation among datum, one-dimensional histograms can provide poor estimation quality. Motivated by this, we propose a new approach to perform more accurate selectivity estimation, even for correlated data. To deal with the correlation that may exist among datum, we adopt well-known techniques in data mining and extract attribute values that occur together frequently using frequent itemsets mining. Through experimentation, we found that our approach is effective in modeling correlations and that this method approximates intermediate relations more accurately. In fact, it gives precise estimates, particularly for the correlated data.

KEYWORDS: Query optimization, Correlated data, Database management system, Frequent itemsets

ARTICLE INFO: Received 12 January 2015, Revised 13 February 2015, Accepted 13 February 2015.

1. Introduction

Among several components in a database management system (DBMS), a query optimizer takes a critical role for overall performance in processing a query. It generates many possible query plans for a given query and chooses the cheapest one [1, 2] in terms of the number of disk accesses in cost-based optimization. To estimate the approximate cost of each query plan,

[*]Corresponding author is with the Division of Computer Science and Engineering, Hanyang University, 222 Wangshimni-ro, Seongdong-gu, Seoul, 133-791, KOREA.
E-mail address: lee@hanyang.ac.kr

a query optimizer needs information on data distribution, and the DBMS typically makes use of a statistical synopsis. Due to the limited time and space, most of current commercial DBMSs use histograms as synopses with AVI (Attribute Value Independence) assumption. Under AVI assumption, attributes in a relation are assumed to have independent data distribution from one other and joint data distribution can be obtained by multiplying individual distributions [3]. However, often, datum are correlated with each other in real world. For example, taller people tend to be heavier than the shorter. People with higher education have a tendency to earn more money. The correlation between attributes affects the distribution of data. If we ignore these characteristics of data and simply use the AVI assumption for estimating cost, a query optimizer can make very poor decisions.

Motivated by this, the paper considers a method for computing high-quality selectivity estimates, even over correlated data. In our approach, we adopt one of the well-known techniques from data mining to build a model for data sets. We use Frequent Itemsets(FI) [4, 5] to capture the correlation among data and estimate the result size of selection and join operations.

Rest of the paper is organized as follows: In Section 2, we discuss related work including prior research. Section 3 describes how frequent itemsets can be used for selectivity estimation in the proposed method. We detail extensive experimentation using TPC-R benchmark and results in Section 4 and the paper is concluded in Section 5.

2. Related Work

2.1 Approaches for Selectivity Estimation

When calculating costs, a query optimizer totally relies on profiles which contain a statistical summary of a relation such as the number of tuples, the number of attribute values and etc., rather than using the real relations [6]. Therefore, profiles play significant roles in query optimization. The oldest and the most common form of profile is a histogram [1, 7]. To build a histogram in DBMS, the domain of attribute values for a single attribute is partitioned into buckets after being sorted, and the histogram keeps the minimum and maximum attribute values of in each bucket. Every bucket has the sum of frequencies of attribute values, as well. Since a histogram does not store all frequencies for values in a bucket, the frequency of every value in a bucket is assumed to be equal to the average of frequencies of all values in that bucket. This is known as the uniform distribution assumption [7, 9]. The advantages of using histograms for query optimization are that there is little run-time over head, that they are inexpensive to store, maintain and compute and that they may give low-error estimates [3,10]. Although various classes of histograms have been proposed, it is known that only some of fundamental approaches are used in practice: equi-width histograms, equi-depth histograms and end-biased [1,11]. Besides these traditional partition-based histograms, one interesting type of histograms is wavelet-based histograms [1,12,13,14]. In this approach, a

mathematical function is used for decomposition process and through this process, original datum are transformed and compressed into a set of numbers, wavelet coefficients. Decomposition processes are repeated until there is a single coefficient, which is called as wavelet decomposition. The wavelet-based histogram shows better accuracy than traditional histograms and most of all this can be naturally extended to the multi-dimensional distribution in the course of wavelet decomposition and reconstruction. Yet, the maintenance for any change of data distribution is much more difficult than partition-based histograms.

Brown and Haas [15] suggest a "data-driven technique" where functional dependencies between attributes are automatically discovered to make an improvement in query optimization. A fuzzy algebraic constraint and other useful relationships are found by BHUNT methodology if there is a correlation between data. First, BHUNT generates candidates that satisfy an algebraic constraint and then, it constructs algebraic constraints for each candidate. Statistical histogramming, segmentation, or clustering techniques are used for this construction. During query processing, the optimizer uses constraints to discover more efficient access paths [16]. In the sense that they try to use a data mining technique to get the good query evaluation, their approach may be closely related to ours.

2.2 Data Mining and Frequent Itemsets

One of the often-mentioned examples for data mining is market basket data [4, 5]. For example, when men go to the market to buy diaper, they likely purchase beer, as well. By finding this correlation between the item diapers and beer among market basket data, we can generate a association rule such that if there is a diaper purchased, then beer is also sold. This rule is denoted as "diaper → beer" [5, 13].

Among the total transactions, if some items are found together frequently, these are called as a frequent itemset (FI) and if the number of items in an FI is k, then it is denoted as k-size itemset. To Mining frequent itemsets is one of the interesting challenges in data mining area and many approaches have been suggested.

3. Cardinality Estimation Using Frequent Itemset Mining

3.1 Overall Description

For the purpose of selectivity estimation using frequent itemsets, we use the famous apriori [4] algorithm. As a measurement to judge if items are eligible as an element of frequent itemset, support and confidence are used. While support is a fraction of the transactions that contains all items to the total transaction, confidence is the ratio of the number of transactions that have all items to the number of transactions that include items in the antecedent.

One important property of frequent itemsets is the upward closure property, which means if an itemset is a frequent itemset with support which is higher than or equal to the threshold, its all subsets are also frequent itemsets. In our approach,

if there is an attribute value set that satisfies a predetermined frequency threshold, we regard this as a frequent itemset and keep the item values in this set with the frequency.

A high-level description of the proposed approach to estimate cardinality is as follows: we begin to mine frequent itemsets among all attribute values using the Apriori algorithm. After finding out all frequent itemsets over a relation, proposed approach builds an FI tree with these itemsets. The FI tree is used as a reference of the data distribution of a relation to estimate the cardinality of the result of a relational algebra operation over the relation, with a few other simple statistics on the relation.

3.2 Data Structure of an FI Tree

Unlike one-dimensional histograms constructed along single attribute, an FI tree exists over a relation. So, there is only one FI tree for a relation while a histogram file exists for an attribute. The root of an FI tree has the cardinality of the relation and other nodes keep item information, such as attribute name, value, a support as a frequency of the itemset and the pointers to its children. The level of the tree implies the size of an itemset, i.e., all nodes at level 1 are the 1-frequent itemsets. Every attribute value on each node on FI tree represents an itemset with its ancestor. One thing to be aware with this tree is that the supports that are on the nodes deeper than level 1 do not mean the support of the item itself. Rather, those are the supports for itemsets that contain all items on their ancestor nodes as well as that node.

<Figure 1> shows an FI tree. N1 at level 1 and N2 at level 2 on the FI tree contain item ‘c3’ and the supports of each are 50% and 30%, respectively. Here, the support of N2 is not for an item,‘c3’, but for the itemset, {‘a1’,‘c3’}. Since the itemset {‘a1’} has a child node of ‘c3’ for the itemset {‘a1’, ‘c3’}, to avoid duplicity, N1 does not need to have a child node of ‘a1’ for that itemset on our FI tree.

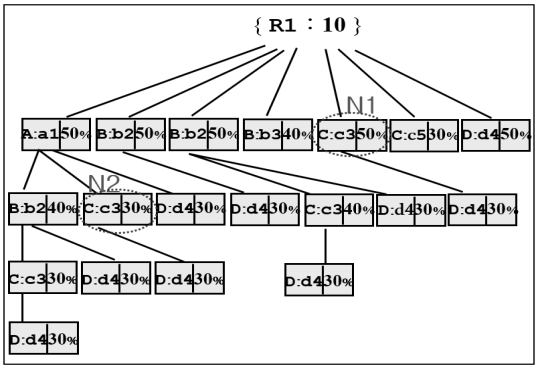

그림 1. FI 트리 예
Figure 1. An example of an FI tree

3.3 Selectivity Estimation for Equality and Join Selection

To estimate the size of a result relation from a selection operation using our profile, we first consider all predicates presented in the selection operation. If all the values which satisfy the predicates are frequent itemsets, we can estimate the exactly same cardinality for result relation since the FI tree keeps the actual frequencies of frequent items. For example, if the selection predicates are R.A=‘a1’ and R.D=‘d4’ in a given

query and {‘a1’, ‘d4’} is a frequent itemset, we can then get the exact cardinality for the result by directly acquiring the frequency from the FI tree. In addition to the FI tree that contains the cardinality of the relation, we also store the number of attribute values for non-frequent items. If we cannot find any attribute value in predicates on the FI tree, we use those additional counts to estimate the frequencies of tuples containing these non-frequent items using standard heuristic methods. Taking the product of the frequencies of frequent itemsets and non-frequent itemsets, finally we approximate the frequency for the predicates in the selection query. Let’s consider the steps described above for the equality selection over the following query plan in <Figure 2>. For the equality operation S1, the predicate set is {‘a1’,‘b1’,‘d4’}. Assume that the cardinality of R is 10 and the value count for attribute B is 6. On the FI tree in <Figure 1>, two attribute values on B exist and the summation of the frequencies for those frequent items is 90% while the frequency of {‘a1’,‘d4’} is 30%. We can see that for the non-FIs, there are 4 attribute values on B and each frequency is assumed as 5% evenly. Therefore, the final frequency for S1 is 1.5% and the estimated cardinality is 0.15 tuples.

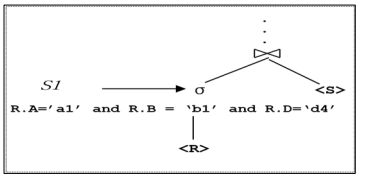

그림 2. 쿼리 계획 예

Figure 2. A query plan tree for the example of the selection

After computing the cardinality of the intermediate relation, we build a new FI tree for that. The query optimizer will use this new FI tree as a profile of the result relation when it processes the further operation where the result relation is involved. In <Figure 2>, after finishing estimation of the equality operation on R, an FI tree for S1 will be created and used for the next join operation.

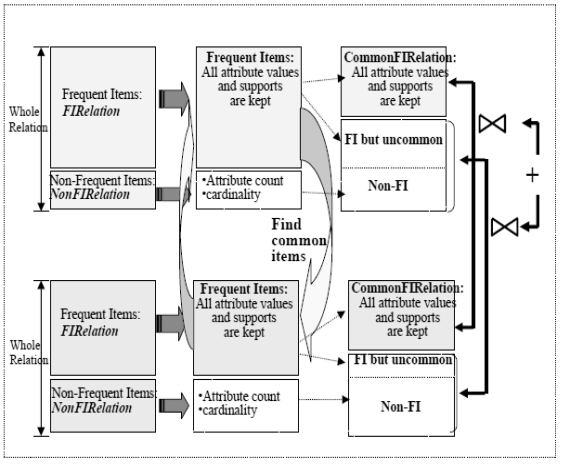

그림 3. 조인 모델링

Figure 3. Join Modeling

Joins are processed in a similar fashion. For join operation, we first need to rebuild the FI trees of the two input relations so that the attribute values belonging to the joined attribute in a query are found at level 1 in the tree. For example, if an FI set of a relation R1 is {‘b2’, ‘d1’, ‘c3’, ‘a1’} and joining attribute is A, then we put {‘a1’} at level 1 and {‘b2’,‘d1’,‘c3’} becomes a child node of {‘a1’} in the tree. After rebuilding the two FI trees using this process, we check all items at level 1 and get the common join attribute values across both two FI trees. If there is an attribute value from the joined attribute

that is present in one relation's FI tree but not the other, we drop that node as well as its all branches from the trees. By doing this, we have only common items for joining attributes on both trees and can easily calculate the frequencies for the relation resulting from joining these values together. In case that the values on nodes are common values but not frequent items, these are deleted from the trees. To deal with these cases, we use standard heuristic methods [17] under the uniform distribution assumption for each pair of attribute values. Just like for equality operation, the product of two frequencies of the attribute values on the FI tree and non-FI tree is the final frequency for the resulting relation. <Figure 3> depicts all procedures for joining two relations.

4. Experimental Results

4.1 Experimental Setting

In our experiments, we compared three different methods for selectivity estimation: 1)Equi-depth histograms, 2)FI trees and 3)Complete histograms. Since the number of buckets in histogram method is a factor that can make a difference in estimation accuracy, we used same bucket size over all equi-depth histograms, where total file size for these synopses was set to be as close to 10M as possible. For FI trees, we also used same profile size, which is 10MB over all relations. We tried to find the threshold which makes the size of a frequent itemset become almost same with the sum of sizes of histograms related with the relation. The third option which is referred as complete histograms, keeps every attribute value and its frequency in different buckets. The number of buckets equals the number of attribute values and we can get actual distribution for every attribute value with complete histograms. If there is no constraint on the size of a profile, this would be the best possible histogram among all one-dimensional histograms. However, in reality, this type of histograms is not usually practical since the purpose of such profiles in database is to model the data distribution from abundant datum. Nevertheless, the reason we used these histograms in our experiments is to check whether even the most ideal histogram can provide good accuracy when data attributes are correlated with each other and whether our method is comparable with the most ideal histogram.

The datum created by TPC-R data generation program [18] is uniformly distributed on all attributes, which is not realistic. So, to get skewed data, we downloaded the publicly available program from researchers at Microsoft [19] and generated more realistic data using their program. This program takes the Zipfian parameter for the degree of skew and uses the Zipfian distribution to provide skew in data. The Zipfian value as a parameter can be from 0 to 4. The parameter value, 0 generates a uniform distribution for each attribute, whereas 4 generates a highly skewed data. By using Zipfian value 2, we created a little skewed data for our experiment. Based this, we prepared two different datasets to see how three methods work for correlated data. The schema of database and query plans were identical, except that we modified some data in database to

introduce additional correlation. We chose 5 queries from the TPC-R benchmark and tested 5 query plans for each query. The accuracy for the estimates from the three different methods have been compared and the formula we used for error rate is as in (1).

$$error\_rate = \frac{|estimated\ \cos t - actual\ \cos t|}{actual\ \cos t} \qquad (1)$$

4.2 Test Result and Discussion

<Table 1> and <Table 2> present the experimental results. The red colored font on these tables denotes the best plan for each method whose estimate is the closest to the actual cost.

By conducting experiments using dataset 1 we could see that when data has little correlation, the Complete Histogram which contain all one-dimensional data distribution information shows the best accuracy as we expected. However, the estimates by the proposed FI tree method show 100 times better than those of equi-depth histograms for most query plans.

For the second data set where some datum are more correlated, FI-tree based method showed the smallest error rate for most query plans, which means proposed approach returns the most accurate estimation. In this environment, the FI method returns even more precise results than Complete Histograms method does. Even though Complete Histograms keep all data value distributions over every attribute, the estimate for multi-dimensional data has to be calculated with the AVI assumption and this caused in less accurate result than FI tree. Therefore, we draw a conclusion that proposed FI tree method outperforms any of one-dimensional histograms in selectivity estimation for correlated data.

표 1. 데이터셋1: 상호연관성이 약한 데이터

Table 1. Dataset1: slightly correlated data

<Query1>

	plan1	plan2	plan3	plan4	plan5	Avg Error Rate
equi-depth H	79.396	20.157	78.819	79.130	46.836	60.87
FI	0.530	0.086	0.522	0.518	0.280	0.39
Complete H	0.007	0.001	0.007	0.000	0.000	0.003

<Query2>

	plan1	plan2	plan3	plan4	plan5	Avg Error Rate
equi-depth H	39.411	7.960	25.142	9.057	10.806	18.48
FI	0.072	0.112	0.063	0.300	0.447	0.19
Complete H	0.005	0.121	0.085	0.306	0.487	0.20

<Query3>

	plan1	plan2	plan3	plan4	plan5	Avg Error Rate
equi-depth H	597.180	609.966	596.337	68.960	490.425	472.57
FI	0.518	0.519	0.520	0.535	0.484	0.51
Complete H	0.002	0.007	0.002	0.052	0.047	0.022

<Query4>

	plan1	plan2	plan3	plan4	plan5	Avg Error Rate
equi-depth H	357.749	640.466	640.348	530.899	1,372.05	708.30
FI	0.032	0.069	0.069	0.734	1.926	0.57
Complete H	0.036	0.083	0.083	0.681	1.884	0.55

<Query5>

	plan1	plan2	plan3	plan4	plan5	Avg Error Rate
equi-depth H	8,030.03	4,185.09	62.072	629.358	760.616	2733.43
FI	27.684	15.175	0.235	2.431	2.705	9.65
Complete H	27.970	15.146	0.145	2.351	2.489	9.62

표 2. 데이터셋2: 상호연관성이 많은 데이터

Table 2. Dataset2: more correlated data

<Query 1>

	plan1	plan2	plan3	plan4	plan5	Avg Error Rate
equi-depth H	0.663	1.490	0.663	3.728	3.911	2.09
FI	0.002	0.001	0.532	0.225	0.026	0.16
Complete H	0.883	0.776	0.882	0.669	0.628	0.77

<Query 2>

	plan1	plan2	plan3	plan4	plan5	Avg Error Rate
equi-depth H	19.966	8.213	17.572	8.984	11.816	13.31
FI	0.402	0.102	0.339	0.248	0.407	0.2996
Complete H	0.434	0.111	0.351	0.246	0.447	0.3178

<Query 3>

	plan1	plan2	plan3	plan4	plan5	Avg Error Rate
equi-depth H	47.990	46.004	47.718	2.261	24.352	33.67
FI	0.441	0.445	0.043	0.281	0.475	0.34
Complete H	0.541	0.571	0.541	0.752	0.748	0.63

<Query 4>

	plan1	plan2	plan3	plan4	plan5	Avg Error Rate
equi-depth H	146.101	169.110	169.055	507.066	1,261.875	450.64
FI	0.528	0.686	0.686	0.619	1.647	0.83
Complete H	0.501	0.646	0.645	0.883	2.253	0.99

<Query 5>

	plan1	plan2	plan3	plan4	plan5	Avg Error Rate
equi-depth H	2.917	3.096	57.677	24.944	461.466	110.02
FI	0.983	0.981	0.357	0.837	1.469	0.9253
Complete H	0.981	0.979	0.217	0.812	1.669	0.9314

5. Conclusions

The performance and efficiency of a DBMS can be significantly affected by the quality of the plan chosen by the query optimizer. Hence the query optimizer is arguably the most important component of a DBMS for query execution time. In this paper, we proposed a novel method to get more accurate selectivity estimation for correlated data during query optimization process in DBMS. We constructs an FI tree after mining frequent itemsets, and used this as a synopsis.

Experimental results show that the FI tree method outperforms equi-depth histograms which are commonly used for query optimizer in commercial DBMS in estimating selectivity. By having smaller than 1 average error rate for most test cases, proposed approach has approved that it can be the potential utility for more effective selectivity estimation.

References

[1] G. Cormode, M. Garofalakis, P. Haas, and C. Jermaine, *Synopses for massive data:samples, histograms,wavelets,sketches,* Foundations and Trends in Databases, Vol. 4, 2012.

[2] R. Kaushik, J. Naughton, R. Ramakrishnan, and V. Chakaravarthy, *Synopses for query optimization:A space-complexity perspective*, ACM Transactions on Database Systems, Vol. 30, No. 4, pp. 1102–1127, 2005.

[3] V. Poosala and Y. Ioannidis, *Selectivity estimation without the attribute value independence assumption,* The 23rd Conference on Very Large Data Bases, pp. 486-495, 1997.

[4] R. Agrawal, and R. Srikant, *Fast algorithms for mining association rules in large databases*, The 20th International Conference on Very Large Data Bases, pp. 487–499, 1994.

[5] S. Brin, R. Motwani, and C. Silverstein, *Beyond market baskets: generalizing association rules to correlations*, ACM SIGMOD International Conference on Management of Data, pp. 265–276, 1997.

[6] N. Bruno, and S. Chaudhuri, *Exploiting statistics on query expressions for optimization*, The ACM SIGMOD, Madison, WI, USA, pp. 263-274, 2002.

[7] P. Haas, I. Ilyas, G. Lohman, and V. Markl, *Discovering and exploiting statistical properties for query optimization in relational databases: A survey, Statistical Analysis and Data Mining*, vol. 1, No. 4, pp. 223–250, 2009.

[8] Y. Matias, and D. Urieli, *Optimal workload-based weighted wavelet synopses, Theoretical Computer Science*, Vol. 371, No. 3, pp. 227–246, 2007.

[9] J. Gryz, and D. Liang, *Query selectivity estimation via data mining, The international conference on intelligent information processing and web mining*, pp. 29-38, 2004.

[10] G. Cormode, A. Deligiannakis, M.Garofalakis, and A.McGregor, *Probabilistic histograms for probabilistic data*, PVLDB, Vol. 2, No. 1, pp. 526–537, 2009.

[11] D. Fuchs, Z. He, and B. Lee, *Compressed histograms with arbitrary bucket layouts for selectivity estimation, Information on Science*, Vol. 177, No. 3, pp. 680–702, 2007.

[12] K. Chakrabarti, M.Garofalakis, R.Rastogi, and K.Shim, *Approximate query processing using wavelets*, The 26th International

Conference on Very Large Data Bases, pp. 111, 2000.

[13] A. Deligiannakis, M. Garofalakis, and N. Roussopoulos, *Extended wavelets for multiple measures*, ACM Transactions on Database Systems, Vol. 32, No. 2, 2007.

[14] A. Deligiannakis, M. Garofalakis, and N. Roussopoulos, *Extended wavelets for multiple measures*, ACM Transactions on Database Systems, Vol. 32, No. 2, 2007.

[15] P. Brown, and P. Haas, *BHUNT: Automatic discovery of fuzzy algebraic constraints in relational data*, The 29th International Conference on Very Large Data Bases, p. 668, 2003.

[16] C. Jermaine, *Playing hide-and-seek with correlations*, KDD, Washington, DC, USA, pp. 559-564, 2003.

[17] J. Ullman, and J. Widom, *Database system implementation,* Prentice Hall, 2000.

[18] TPC benchmark R. Decision Support, http://www.tpc.org

[19] Program for TPC-D Data Generation, ftp://ftp.research.microsoft.com/users/viveknar/tpcdskew

빈발항목 마이닝을 이용한 선택도 측정

엄보윤[1], Christopher Jermaine[2], 이춘화[3]

[1]*한국전자통신연구원 지능형융합미디어연구부*

[2]*라이스대학교 전산학과*

[3]*한양대학교 컴퓨터공학부*

요 약

쿼리 최적화기에 의해 선택된 쿼리 계획은 전체 쿼리 실행 속도에 지대한 영향을 미치기 때문에, 데이터베이스관리시스템의 쿼리 최적화 기능은 쿼리 처리 과정에 있어 중요하다. Cost 기반 최적화에서 쿼리 최적화기는 모든 가능한 쿼리 계획들의 비용을 데이터베이스의 데이터 분포 정보를 기반으로 추정한다. 일반적으로 상용화되고 있는 DBMS에서 가장 흔하게 사용되는 데이터 분포 통계 정보는 히스토그램 방식으로 구축된 형태이다. 그러나 각각의 데이터들에 상호 연관성이 있는 경우, 일차원 히스트그램 방법은 형편없는 추정치를 계산해낸다. 본 논문에서는 쿼리 최적화 과정에서 보다 정확한 쿼리 비용을 계산하여 쿼리 계획을 선출할 수 있도록 하기 위해 데이터마이닝의 기술 중 하나인 빈발항목 마이닝(Frequent Itemsets Mining) 방법을 적용하였다. 실험을 통해 제안하는 방법이 상호 연관관계 있는 데이터들에 있어 히스토그램보다 좋은 추정치를 보여 줌을 확인하였다.

Acknowledgments

This work was supported by the ICT R&D program of MSIP/IITP. [2014-044-042-001, Development of Open Screen Service Platform with Cooperative and Distributed Multiple Irregular Screens] and by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education(No. 2013R1A1A2007616).

Boyun Eom was born in Cheongju in South Korea. She received her M.S. degree in computer & information science & engineering from University of Florida, Gainesville, US, in 2005. Currently, she is a senior researcher at ETRI. Her research interests include cloud computing, smart home, database, and middleware and services computing technology.

E-mail address: eby@etri.re.kr

Christopher Jermaine is a associate professor in the Division of Computer Science at Rice University. He received an M.S. in computer science and engineering at Ohio State University, and a Ph.D. from the College of Computing at Georgia Tech. His research interests include database, data analytics and biomedical data.

E-mail address: cmj4@rice.edu

Choonhwa Lee is an associate professor in the Division of Computer Science and Engineering at Hanyang University, Seoul, South Korea. He received his B.S. and M.S. degrees in computer engineering from Seoul National University, South Korea, in 1990 and 1992, respectively, and his Ph.D. degree in computer engineering from the University of Florida, Gainesville, in 2003. His research interests include cloud computing, peer-to-peer and mobile networking and computing, and services computing technology.

E-mail address: lee@hanyang.ac.kr