Horizontal partitioning method for test verification in parallel database systems

Feras Ahmad Hanandeh

Department of Computer Information Systems,
Faculty of Prince Al-Hussein Bin Abdallah II
for Information Technology,
Hashemite University,
Jordan
Email: feras@hu.edu.jo

Abstract: In parallel database systems the partitioning methods considered in current researches are static. This research paper presents a partitioning method to divide the database relations into dynamic horizontal partitions. Every partition contains some tuples of the database relation. These partitions will be checked using the subsets generated from the integrity constraints during the test verification process. Furthermore, the integrity of these partitions will be enforced by the generated integrity rules. It presents the algorithm of distributing the integrity test(s) among the horizontal partitions. As our intention is to parallelise the execution of the integrity tests, the relations specified in the test are dynamically partitioned into several parts based on the number of available processors. Each processor checks the validity of the test concurrently by accessing the partition assigned to it.

Keywords: parallel databases; distributed database; fragmentation; integrity test generation; integrity maintenance.

Reference to this paper should be made as follows: Hanandeh, F.A. (xxxx) 'Horizontal partitioning method for test verification in parallel database systems', *Int. J. Advanced Intelligence Paradigms*, Vol. X, No. Y, pp.000–000.

Biographical notes: Feras Ahmad Hanandeh is currently an Associate Professor at the Prince Al Hussein Bin Abdullah II Faculty of Information Technology, Al-Hashemite University, Jordan. He obtained his PhD in Computer Science from University Putra Malaysia, Malaysia in 2006. His current research interests include distributed databases, parallel databases focusing on issues related to integrity maintenance, transaction processing, query processing and Optimization; grid computing, artificial intelligence and Geographical information systems.

This paper is a revised and expanded version of a paper entitled [title] presented at [name, location and date of conference].

Comment [t1]: Author: If a previous version of your paper has originally been presented at a conference please complete the statement to this effect or delete if not applicable.

1 Introduction

Depending on the actual SQL statement, some databases may use static or dynamic partitions. Static partitions occur at compile-time, with the information about the partitions accessed beforehand. Dynamic partitions occur at run-time, meaning that the

exact partitions to be accessed by a statement are not known beforehand. A sample scenario for static partitions is an SQL statement containing a WHERE condition with a constant literal on the partition key column. An example of dynamic partitions is the use of operators or functions in the WHERE condition.

Parallel database systems are used to process a unit of execution (a transaction) in a parallel manner. That is, a transaction can be executed by multiple networked computers or processors in a unified manner.

In distributed database systems, data fragmentation is implemented to split a relation into logically related and correct parts. A relation can be fragmented in two ways: the first is horizontal fragmentation which is a horizontal subset of a relation that contains those of tuples which satisfy selection conditions. The second is Vertical fragmentation which is a subset of a relation which is created by a subset of columns.

Integrity constraint maintenance usually goes through several phases starting from declaring constraints (known as constraint specification) i.e., stating declaratively what constitutes are semantically allowed for database state and also what should be done if constraints are invalidated (Grefen, 1992; Martinenghi, 2005). The next phase is concentrated on investigating the generated set of constraints together to guarantee the consistency.

The process of maintaining the integrity in parallel databases is different from the process used in centralised and distributed databases in which the database relations are horizontally partitioned. Intervention of the user when he/she decides to abort the update at commit time without repair action(s) in the event of constraint violation is also inefficient since rollback and recovery must occur at all partitions which are affected by the update operation. Thus, this research focuses on proposing an approach for horizontal partitioning. This approach is mainly used for partitioning parallel database relations for test verification process. The tuples in each partition are indexed according to the test attribute of the generated integrity subtests. So each processor will access a specific partition.

The tuples in the database relations are required to be settled properly for test verification using more than one processor. For the settlement of tuples in a parallel database, a static rule partitioning method is presented in McCarroll (1995). Based on this method, checking is performed by accessing the tuples in a sequential manner. Therefore, more time is required to perform the test.

This research paper considers the algorithm of generating simplified integrity tests presented in Hanandeh et al. (2013) which could be used to automatically verify that database updates do not introduce any violation of integrity. Throughout this research the example job agency database is used, as given in Table 1. This example is taken from Wang (1992) with some modification.

Table 1 The relations of the job-agency database

Relations	*Intended meaning*
Person (pid, pname, placed)	<p, m, d> ∈ Person, p is the id of the person, m is the name of the person and his status whether placed or not is d
Company (cid, cname, totsal)	<c, e, t> ∈ Company, c is the id of the company's name e with total salary t
Job (jid, jdescr)	<j, r> ∈ Job, j is the id of the job description r

Table 1 The relations of the job-agency database (continued)

Relations	*Intended meaning*
Placement (pid, cid, jid, sal)	<p, c, j, s> ∈ Placement, person p is hired in company c for job j with salary s
Application (pid, jid)	<p, j> ∈ Application, person p applies for job j
Offering (cid, jid, no_of_places)	<c, j, k> ∈ Offering, the company c is offering k places for job j

The remainder of this paper is as follows. In Section 2, we briefly review some of the related work materials. In Section 3, the partitioning strategy used in this research is discussed. Section 4 presents the results and evaluation. Section 5 presents the test verification process by distributing integrity tests according to the new partitioning strategy used in this research. Finally, we conclude in Section 6.

2 Literature review

Simon and Valduriez (1984) have proposed a method for using partitioning and placement information to distribute global constraints into fragment constraints using SABRE distributed database system. The aim is to minimise inter-site traffic. One of the sites is the Query Master Site, which broadcasts details of the updates to other sites at which localised constraints need to be checked. Their method minimises inter-site traffic but fails to handle partitioning strategies of general complexity.

Qian (1989) has presented a method to avoid expensive access to non-local data to minimise the overheads and the complexity of distributing integrity testing at execution time. His method uses partitioning and placement information to derive fragment constraints from global constraints defined on a distributed database. This enables updates to each site to be tested for safety without the need for expensive access to non-local data.

McCarroll (1995) used a theorem proving method for deriving sufficient and complete tests at schema compile-time to ensure that database state will not be violated when the transaction is executed with respect to a constraint. McCarroll (1995) used distributed tests to reduce the amount of data needed to be accessed in integrity enforcement. His approach has parallelised the integrity tests. Specialised run-time support will be needed for effective use of distributed tests. The optimal selection of tests from alternatives also assumes some sophisticated run-time decision making algorithm. The novel feature of the method proposed by McCarroll is the automatic division of integrity tests into subtests, which independently investigate the safety of changes to disjoint sections of a database that is affected by the update transaction. Here, parallelism is exploited in addition to the minimisation of the amount of data accessed by integrity enforcement activity. In addition, there will be no hindrance for the testing to commence. According to McCaroll's method the cost of transporting data among several nodes of the database is significant. Such cost is not a critical factor in parallel databases but is critical in distributed databases (Alwan et al., 2010). Now that there are more parallel databases in existence, it might be worth looking closely at their architectures and trying to see how to generate integrity tests, which exploit parallelism better.

The distributed nature of the current large and complex organisations involves a very large number of transactions from their customers/clients (Wen et al., 2011). These data will be accessed and modified from different users at different physical places. Clearly, the integrity of data is essential for any business. To preserve the database integrity, several constraints (conditions) must hold on all valid database states. In this regard, all transactions (insert, update and delete) have to be verified according to a predefined constraint during the run time. Verification process against large databases (Kim et al., 2011) is extremely difficult and takes a long time that negatively affects the efficiency of executing these valuable business transactions.

In Madiraju et al. (2006), a general algorithm was presented for checking global semantic integrity constraints in an XML setting. The presented approach does not require update statement to be executed before the constraint check is carried out and hence any rollback situations are avoided. It achieves speed as the sub constraint checks are executed in parallel. Simple classes of integrity constraints were examined in Grefen (1992), where the parallel constraint checking was considered in this scheme. A limited static integrity constraints algorithm is presented in McCarroll (1995). Furthermore, there is a need to write complex transition axioms to describe the update operations, which is time-consuming and less efficient.

Salwani et al. (2011) present attribute reduction by finding the minimal attribute from a large set of attributes a limited algorithm to the semantic integrity constraints is presented in Madiraju and Sunderraman (2004). Soumya et al. (2008) proposed a technique for relational databases to achieve optimisation of constraint checking process in distributed databases by taking advantage of the parallelism environment, compile time constraint checking, localised constraint checking, and the history of constraint violations. The architecture mainly consists of two modules: constraint analyser and constraint ranker for analysing the generated constraints and for ranking them.

In Matsui and Matsuo (2010), a distributed optimisation algorithm is proposed. An interval partitioning algorithm to solve the continuous constraint satisfaction problem is proposed in Sekhar et al. (2011). The method comprises a dynamic tree search management system that also invokes local search in selected subintervals. Ramification problem refers to the production of indirect effects after the execution of an update in order to ensure the satisfaction of integrity constraints is proposed in Papadakis et al. (2011).

Lena (2014) presents an intelligent fragmentation and replication approach for a distributed database system. With this approach, cloud storage can be enhanced with a semantically-guided flexible query answering mechanism that will provide related but still very relevant answers for the user. The approach combines fragmentation based on a clustering with data replication.

And finally, Rimma and Nicolas (2011) put forward a partitioning advisor that recommends the best partitioning design for an expected workload. Their tool recommends which tables should be replicated (i.e., copied into every compute node) and which ones should be distributed according to specific column(s) so that the cost of evaluating similar workloads is minimised.

3 Parallel database partitioning method

As the generated integrity test will be executed at run-time, the relations will be partitioned according to a specific criterion to be investigated using the generated integrity test. The way of partitioning the relations affects the execution of the integrity test. In this research work, a horizontal method is proposed to partition the relations for parallel database systems. The horizontal method concentrates in generating horizontal partitions of a relation at any time, consisting of groups of tuples based on the generated integrity test(s).

The number of processors determines the number of horizontal partitions during the test verification process. The number of horizontal partitions and the number of tuples in the relation determine the number of tuples in each partition. The horizontal partition will be emitted in different times $t1$, $t2$ such that $(\forall t1 \forall t2)(t2 > t1)$. The horizontal method is applicable to parallel database systems that work in the architecture of type nothing share.

During the test verification process, the database relations will be split into several partitions and these partitions will be accessed concurrently by the available processors. Partitioning the database relations is a logical process which indexes the database relations and then group the relations into several horizontal partitions based on the generated integrity test. The flexibility of creating the horizontal partitions at any time is considered one of the major benefits of the horizontal method. This is because each processor will be given the same or approximate number of tuples to be evaluated. This will ensure that each processor is given the same amount of workload. Also this method is not restricted to a specific physical or static partitioning. The accessing method used is important to determine the time taken to access the requested tuples based on the generated test. Since each partition has been indexed according to the condition as stated in the test, therefore searching for the tuples that satisfy the condition of the test is easier and faster as compared to the partition without indexing.

The proposed method partitions a relation based on the number of available processors during the test verification process. The main processor is responsible of sending the subtest to all the other processors. Therefore, the main processor will not be involved in the test verification process. The number of tuples, K, in each partition is calculated as, $K = \lfloor (T / (\text{number of processors} - 1)) \rfloor + 1$, where T is the number of tuples of a relation which will be verified by the test.

Test attribute (TA) is the attribute of the generated test that is substituted with the values of the submitted update operation. The generated test may have more than one TA when two or more attributes are being substituted.

The tuples in each partition are indexed based on the TA. A pointer, j, points to the first tuple of each horizontal partition. The creation of the horizontal partitions in this approach is defined as follows:

```
CREATE HORIZONTAL PARTITION HPname
ON P
{
RANK (TA)
        j = 1
```

```
        Do while j < T
        {
                σ Between {j, K*i} (P) → partition[i]
                j = j + K
                i = i + 1
        }
        if j > T
        σ {j, T}(P) → partition[i];
}
```

for i ∈{1, 2,…,n}, where n is the number of available processors not including the main processor.

where HPname is the name of the horizontal partition and *P* is the name of the relation. The tuples of each partition of the relation, *P* will be indexed based on the TA. $Proc_r - 1$ is the number of processors which are responsible for checking the integrity subtests.

Example: Suppose that there are 130 tuples in the *Placement* relation, 50 tuples in the *Job* relation and five processors are available to verify the following generated tests:

$$comp\left(T^{exp}.\text{Exp}\right) \leftarrow \neg \text{Job(j, 'Director')}$$
$$\vee(\exists u \exists p \exists q)\left(\text{Placement(u, c, p, q)} \wedge \text{Job(p, 'Senior Secretary')}\right)$$

$$suff\left(T^{exp}.\text{Exp}\right) \leftarrow (\exists w \exists z)\left(\text{Placement(w, c, j, z)}\right)$$

These complete and sufficient tests are generated for the constraint (C) $(\forall w \forall x \forall y \forall z \exists u \exists p \exists q)(Placement(w, x, y, z) \wedge Job(y, `Director') \rightarrow Placement(u, x, p, q) \wedge Job(p, `Senior\ Secretary'))$ and the update operation insert ($Placement(e, c, j, s)$). For the complete test, two relations are involved: *Job* and *Placement*. The TA for the *Job* relation is *jid* (substituted with the value *j*) and the TA for the *Placement* relation is *cid* (substituted with the value *c*). Two horizontal partitions are generated. The first horizontal partition will be created based on the *jid* attribute, which will partition the Job relation. The number of tuples in each partition is calculated as, $K = \lfloor 50 / 4 \rfloor + 1 = 13$. The first horizontal partition, Hp_1_C is created as follows:

```
CREATE HORIZONTAL PARTITION Hp_1_C
ON JOB
{
RANK (jid)
        σ Between {01, 13} (JOB) → partition [1]
        σ Between {14, 26} (JOB) → partition [2]
        σ Between {27, 39} (JOB) → partition [3]
        σ Between {40, 50} (JOB) → partition [4]
}
```

According to this horizontal partition, Hp_1_C, the *Job* relation will be indexed based on the *jid* attribute. The first 13 tuples of the *Job* relation are located in partition 1. The second 13 tuples of the *Job* relation are located in partition 2 and the third 13 tuples of the *Job* relation are located in partition 3. The last 11 tuples are located in partition 4.

The second horizontal partition will be created based on the *cid* attribute, which will partition the *Placement* relation. The number of tuples in each partition is calculated as, $K = \lfloor 130 / 4 \rfloor + 1 = 33$. The second horizontal partition, Hp_2_C is created as follows:

```
CREATE HORIZONTAL PARTITION Hp_2_C
ON PLACEMENT
{
RANK (cid)
        σ Between {01, 33} (PLACEMENT) → partition [1]
        σ Between {34, 66} (PLACEMENT) → partition [2]
        σ Between {67, 99} (PLACEMENT) → partition [3]
        σ Between {100, 130} (PLACEMENT) → partition [4]
}
```

According to this horizontal partition, Hp_2_C, the *Placement* relation will be indexed based on the *cid* attribute. The first 33 tuples of the *Placement* relation are located in partition 1. The second 33 tuples of the *Placement* relation are located in partition 2. The third 33 tuples of the *Placement* relation are located in partition 3. The last 31 tuples are located in partition 4.

For the sufficient test, one relation is involved that is *Placement*. There are two TA's for the *Placement* relation, namely *cid* (substituted by the value *c*) and *jid* (substituted with the value *j*). Only one horizontal partition is generated. The horizontal partition will be created based on the *cid* and the *jid* attributes, which will partition the *Placement* relation. The number of tuples in each partition is $K = \lfloor 130 / 4 \rfloor + 1 = 33$. The horizontal partition, Hp_1_S is created as follows:

```
CREATE HORIZONTAL PARTITION Hp_1_S
ON PLACEMENT
{
RANK (cid, jid)
        σ Between {01, 33} (PLACEMENT) → partition [1]
        σ Between {34, 66} (PLACEMENT) → partition [2]
        σ Between {67, 99} (PLACEMENT) → partition [3]
        σ Between {100, 130} (PLACEMENT) → partition [4]
}
```

According to this horizontal partition, Hp_1_S, the *Placement* relation will be indexed based on the two attributes, *cid* and *jid*. For the same *cid* the tuples are indexed based on the *jid*. The first 33 tuples of the *Placement* relation are located in partition 1. The second 33 tuples of the *Placement* relation are located in partition 2. The third 33 tuples of the *Placement* relation are located in partition 3. The last 31 tuples are located in partition 4.

4 Evaluation and results

In order to evaluate the effect of parallel test verification, the two complete and sufficient tests in the previous section were used to verify database relations consisting of 100,000 tuples each. Table 2 lists the average response time during the test verification process using two to four processors and using the dynamic horizontal partitioning method. The average response time is calculated in milliseconds.

Comment [t2]: Author: Please note that 'Table 6.1' has been changed to 'Table 2'.

Table 2 Average response time of performing the two complete tests and the sufficient one on relations consisting of 100,000 tuples involving two to four processors and using the dynamic horizontal partitioning method

Comment [t3]: Author: Please note that 'Table 6.1' has been changed to 'Table 2'.

Number of processors	*Response time in milliseconds Hp_1_C*	*Response time in milliseconds Hp_2_C*	*Response time in milliseconds Hp_1_S*	*Average response time of the three tests in milliseconds*
2	2.174	2.353	1.437	1.988
3	1.98	2.329	1.20	1.836
4	1.742	1.990	.99	1.574

Also the two complete and sufficient tests in the previous section were used to verify database relations consisting of 100,000 tuples each to specify how the database relations are subdivided into smaller individually addressable units. Table 3 lists the average response time during the test verification process using two to four processors and using the static horizontal partitioning method considered in the literature (Grefen, 1992; Martinenghi, 2005; McCarroll, 1995). The average response time is calculated in milliseconds.

Comment [t4]: Author: Please note that 'Table 6.2' has been changed to 'Table 3'.

Table 3 Average response time of performing the two complete tests and the sufficient one on relations consisting of 100,000 tuples involving two to four processors and using the static partitioning method

Comment [t5]: Author: Please note that 'Table 6.2' has been changed to 'Table 3'.

Comment [t6]: Author: Please confirm if this should be 'static **horizontal** partitioning method' instead.

Number of processors	*Average response time of the three tests in milliseconds*
2	3.172
3	2.750
4	2.309

In the above experiments, we compare the static partitioning approach with the proposed dynamic partitioning one. Clearly, in both approaches the average response time of the tests were shorter when using more processors. Considering the proposed dynamic horizontal partitioning method it is very clear that the average response time of performing the mentioned tests is shorter than performing the same tests with the static partitioning approach.

5 Test verification by distributing integrity tests

During run-time, when an actual update is submitted by the end user, the complete tests as well as the sufficient tests are generated by the algorithm presented in Hanandeh et al. (2013). The generated tests are sometimes composed of disjunctive of subtests. The

verification of the subtests is executed in parallel through all the created horizontal partitions according the approach presented in this research paper.

Event condition action (ECA) is a short-cut for referring to the structure of integrity rules in event driven architecture and active database systems.

This integrity rule consists of three parts:

- the event part, which specifies the signal that triggers the invocation of the rule
- the condition part is replace by the generated integrity test (Hanandeh et al., 2013) that, if satisfied or evaluates to true, causes the action to be carried out
- the action part, which consists of updates or invocations on the local data.

The integrity rules will be verified to select those that may violate the update operation. This is done by analysing each rule and checking whether the operation in the Event part has the same operation and relation as in the update operation, then the rule is selected. For each rule, based on the test specified, the partitioning process is performed. In this work, the horizontal partitioning is applied. The parallel database partitioning method splits the database relations into dynamic horizontal partitions. Each partition contains a portion of the database relation. The distribution of the database relations to different portions depends on the test, Test-C where Test-C is either the *comp*(T^{exp}.Exp) or the *suff*(T^{exp}.Exp) test.

6 Conclusions

Centralisation, fragmentation and partitioning of database relations are important to design well and be performed before or during the test verification process. In centralised database systems, the database tables are settled as local to every test being executed. For distributed database systems, the data is physically distributed over the network. The directions of the previous researches were to localise the test verification process. This will reduce the amount of data transfer across the network as well as the number of sites involved. In parallel database systems, the database relations are localised and logically partitioned. The approaches in the literature have considered the static partitioning method to specify how the database relations are subdivided into smaller individually addressable units. Therefore, the partitioning happens before the execution of the checking process. The approach presented in this research work considers the dynamic logical partitioning method. The decision of how the database relations are going to be partitioned for test verification process is done during the run-time process. The approach exploits the generated integrity subtest(s) to decide how the data is going to be partitioned. The TA will be the criterion of the partitioning process. The partitions are indexed based on the TA. The test verification process uses the direct access method. Based on the generated integrity subtest(s) and the dynamic logical partitions, the integrity rules are generated. The action of the integrity rules depends on the condition of the integrity rules, which is one of the integrity subtest(s).

References

Alwan, A., Ibrahim, H. and Udzir, N. (2010) 'A model for ranking and selecting integrity tests in a distributed database', *Int. J. of Inf. Tech. and Web Engineering*, Vol. 3, No. 3, pp.65–84, University of Milan, Italy.

Grefen, P.W.P.J. (1992) *Integrity Control in Parallel Database Systems*, PhD Thesis, University of Twente, Netherlands.

Grefen, P.W.P.J. and Apers, P.M.G. (1991) 'Integrity constraint enforcement through transaction modification', *Proceedings of the 2nd International Conference on Database and Expert Systems Applications (DEXA'91)*, Berlin, (Germany), pp.362–367.

Hanandeh, F., Abdallah, E.E., Abdallah, A.E. and Al-Daoud, E. (2013) 'Simplified approach for generating integrity tests in distributed database systems', *Int. J. Innovation and Learning*, Vol. 13, No. 4, pp.375–387.

Kim, J., Shim, J. and Ahn, K. (2011) 'Social network service: motivation, pleasure, and behavioral intention to use', *J. of Computer Information Systems*, Vol. 51, No. 4, pp.92–101.

Lena, W. (2014) 'Clustering-based fragmentation and data replication for flexible query answering in distributed databases', *Journal of Cloud Computing: Advances, Systems and Applications – A Springer Open Journal*, Vol. 3, No. 1, p.18.

Madiraju, P. and Sunderraman, R. (2004) 'A mobile agent approach for global database constraint checking', *Proc. of the ACM Symposium on Computing*, Nicosia, pp.679–683.

Madiraju, P., Sunderraman, R. and Haibin, W. (2006) 'A framework for global constraint checking involving aggregates in multidatabases using granular computing', *Proceedings of IEEE International Conference on Granular Computing*, Atlanta, pp.506–509.

Martinenghi, D. (2005) *Advanced Techniques for Efficient Data Integrity Checking*, PhD Dissertation, Roskilde University, Denmark.

Matsui, T. and Matsuo, H. (2010) 'A constraint based formalization for distributed cooperative sensor resource allocation', *International Journal of Intelligent Information and Database Systems*, Vol. 4, No. 4, pp.307–321.

McCarroll, N.F. (1995) *Semantic Integrity Enforcement in Parallel Database Machines*, PhD Thesis, Department of Computer Science, University of Sheffield, Sheffield, UK.

Papadakis, N., Christodolou, Y., Sartzetakis, P. and Papadakis, K. (2011) 'Constraint satisfaction in XML temporal databases', *Int. J. of Reasoning-Based Intelligent Systems*, Vol. 3, No. 1, pp.44–58.

Qian, X. (1989) 'Distribution design of integrity constraints', *Proceedings of the 2nd International Conference on Expert Systems*, California, USA, pp.205–226.

Rimma, N. and Nicolas, B. (2011) 'Automated partitioning design in parallel database systems', *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, ACM, New York, NY, USA, pp.1137–1148.

Salwani, A., Laleh, G. and Mohd, Z. (2011) 'Re-heat simulated annealing algorithm for rough set attribute reduction', *Int. J. of the Physical Sciences*, Vol. 6, No. 8, pp.2083–2089.

Sekhar, C., Kumar, A., Csendes, T. and Posfai, J. (2011) 'An interval partitioning algorithm for constraint satisfaction problems', *International Journal of Modeling, Identification and Control*, Vol. 14, Nos. 1–2, pp.133–140.

Simon, E. and Valduriez, P. (1984) 'Design and implementation of an extendible integrity subsystem', *Proceedings of the 1984 ACM SIGMOD International Conference on the Management of Data*, Boston, USA, pp.9–17.

Soumya, B., Madiraju, P. and Ibrahim, H. (2008) 'Constraint optimization for a system of relation databases', *Proc. of the IEEE Int. Conf. on Computer and Info. Technology*, Sydney, pp.155–160.

Wang, X.Y. (1992) *The Development of a Knowledge-Based Transaction Design Assistant*, PhD Thesis, Department of Computing Mathematics, University of Wales College of Ca, 15-18.rdiff, Cardiff, UK.

Comment [t7]: Author: Please cite the reference in the text or delete from the list if not required.

Wen, C., Prybutok, V. and Xu, C. (2011) 'An integrated model for customer online repurchase intention', *J. of Computer Information Systems*, Vol. 52, No. 1, pp.14–23.