Transactional Agents for Pervasive Computing

Machigar Ongtang
Dept. of Computer Science and Engineering, Pennsylvania State University
ongtang@cse.psu.edu

Ali R. Hurson
Computer Science Dept., Missouri University of Science and Technology
hurson@mst.edu

Yu Jiao
Computational Sciences and Engineering Division, Oak Ridge National Laboratory
jiaoy@ornl.gov

Abstract

Pervasive computing enables seamless integration of computing technology into everyday life to make up-to-date information and services proactively available to the users based on their needs and behaviors. We aim to develop a transaction management scheme as a pertinent component for such environment supported by either structured or ad hoc networks. We propose Transactional Agents for Pervasive COmputing (TAPCO), which utilizes a dynamic hierarchical meta data structure that captures the semantic contents of the underlying heterogeneous data sources. Mobile agents process the transactions collaboratively, to preserve ACID properties without violating local autonomy of the data sources. TAPCO is simulated and compared against Decentralized Serialization Graph Testing (DSGT) protocol. The results show that TAPCO outperforms DSGT in several ways. In contrast to DSGT that did not consider local transactions, TAPCO supports both local and global transactions without violating the local autonomy.

1. Introduction

The proliferation of pervasive computing requires a proper infrastructure in place. As all actions performed are in the form of transactions, a suitable transaction management protocol becomes a critical element. To reflect such the need, consider the situation when a group of independent users reside in the same area such as in library or office building. They could be connected via access points or ad hoc connections. As part of pervasive environment, they must be able to share, access, and manipulate data and services. They now face difficulties of how to: (i) know which data are available, (ii) ensure the correctness of their transactions when multiple users simultaneously submit transactions to multiple autonomous data sources, (iii) pervasively make information up-to-date, and (iv) synchronize the information regardless of the constraints imposed by the technology. Data sources are dynamic, ubiquitous and heterogeneous. They may reside in either fixed or mobile network. Mobile devices can be both data sources and consumers. In the past, most of the researches in this area focused on service discovery and query processing. However, mobile applications may involve data manipulation. Our goal is to develop a transaction management protocol for pervasive computing, to proactively allow access and update of dynamic data and resources.

Conventional transaction management protocols are mainly designed for human-computer interaction computing, not for a proactive computing model. Some of the proposed solutions are too restrictive, [1, 2, 4] and/or based on the assumptions that do not realistically address the aforementioned challenges [4, 6]. We propose a novel Transactional Agents for Pervasive COmputing (TAPCO) as the middleware infrastructure for data access and manipulation in pervasive environment. Mobile data sources automatically form a virtual hierarchical structure that facilitate data access and transaction resolution, using semantic-based data clustering technique [11]. Our protocol preserves local autonomy and heterogeneity while conforming to the serializability rules that ensure correctness of all transactions and system integrity. It is a non-locking, pessimistic protocol that supports both compensatable and non-compensatable transactions. Consequently, it avoids the need for global locks and cascading aborts, and thus offers a reduced processing time. Autonomous agents allows parallel processing of global subtransactions, and reduces network traffic. They also allows the transactions to be processed to completion while the mobile client is disconnected.

This paper is organized into 5 sections. Section 2 provides some essential background and presents some related works. Section 3 details our TAPCO algorithm. Simulation and its results are presented in section 4. Finally, section 5 draws the paper to conclusion.

2. Backgrounds

Pervasive environment consists of heterogeneous fixed and/or mobile data sources, receiving transactions from mobile and/or fixed clients. There are two levels of control. At the global level, each global transaction – GT is decomposed into several global subtransactions – GSTs, each to be executed at the local level. Local autonomy and heterogeneity of local data sources require the global transaction manager to support various types of concurrency control schemes, which may be invisible to the global level. The local histories (LH) of the execution order of both local transactions (LTs) and GSTs at local sites are invisible to the global transaction manager that maintains the global history (GH). Two GTs, which otherwise do not conflict, may conflict over LTs - *indirect conflict*. Thus, the transaction management must preserve the following serializability rules: (i) Every LH is conflict serializable, and (ii) For GT_i and GT_j, if an operation of GT_i precedes an operation of GT_j in one LH, all operations of GT_i must precede any operation of GT_j in all LHs. The architecture of such transaction management mechanism is shown in Fig. 1.

The **content-based clustering** reduces the search cost for accessing data in mobile ad hoc network [11]. It extended Summary Schema Model (SSM) originally proposed to facilitate semantic based query/transaction resolution in mobile multidatabase [3, 8]. SSM is a hierarchical meta data structure. Its leaf nodes store part of the schema shared by the local data sources. Higher-level nodes are Summary Schema Nodes (SSNs), providing increasingly abstract view of the underlying data by summarizing the semantic contents of its child nodes. The relationships between terms in the SSM include synonyms, hypernyms (broader terms) and hyponyms (narrower terms). The participating hosts are partitioned into semantic-related groups called *semantic-based clusters* based on the semantic similarity of their data content as determined by the online thesaurus [7]. Small clusters are recursively merged into larger clusters, fusing their data contents to a more general description associated with the larger cluster. This recursive process stops when all participating hosts are merged to a single cluster, and then create the SSM as shown in Fig. 2. For each cluster, a host is selected as the centroid, which is logically the SSN of the SSM hierarchy. The centroid is selected based on hardware characteristics and data contents (how well the centroid represents the cluster's content). The centroid keeps the updated information about the characteristics of all the hosts in the cluster to change the centroid accordingly. In Fig. 2, the centroid of each cluster maintains semantic content of the cluster; the upper-level centroids contain the abstract view of child nodes. TAPCO employs the semantic-based clustering to build a virtual hierarchical structure as its semantic platform for both accessing the data and facilitating the transaction management tasks in pervasive environment.

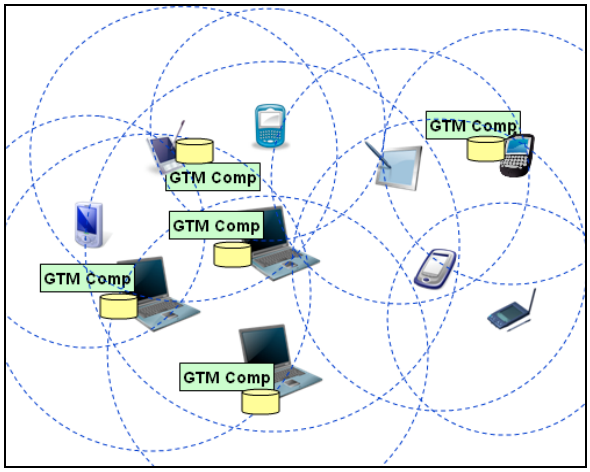

Fig. 1. Transaction Management in Pervasive Computing

2.1 Related Works

Mobile Semantic Serializability scheme proposed in [1, 2] handles the transactions that present a serial execution of independent atomic units called *modules*. It assumes that mobile databases are disjoint semantic entities, i.e., an operation in one global subtransaction does not depend on the data from another global subtransaction; thus, it could simply implement a variant of the conventional two-phase locking protocol. NC-Transaction scheme within MoGATU framework [9, 10] aimed to maintain a neighborhood-based consistency by electing some active witnesses to monitor the transaction to ensure its correctness and fairness. AMOR (Agents, MObility, and tRansaction)

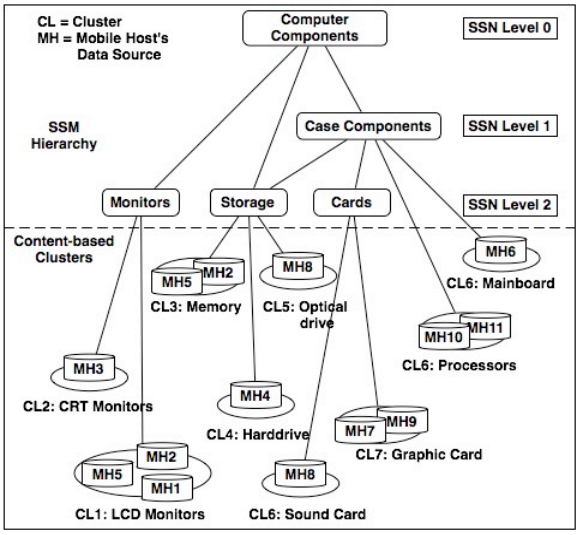

Fig. 2. Semantic-based Clustering

[6] is an agent-based transaction management protocol for peer-to-peer wired environment with Decentralized Serialization Graph Testing (DSGT). AMOR uses Resource Agent to wrap local database, log local service invocations, and local conflicts. Transaction Agents use such information to collaboratively resolve conflicts. It, however, did not address local autonomy and indirect conflicts.

3. TAPCO Scheme

Transaction management for pervasive computing needs to: (i) handle heterogeneous data sources while preserving local autonomy, (ii) conform to the serializability rules mentioned in section 2.1, (iii) work without infrastructure support, and (iv) accommodate the dynamic topology and frequent disconnections. TAPCO has three major functionalities, which address these four challenges: *(1.) Dynamic Semantic-based Clustering* involves the creation of the semantic-based cluster to address the first challenge, as detailed in section 2. In TAPCO, the centroid also informs all nodes in the cluster about the members of the centroid. To clarify our discussion, the centroid refers to the physical host, while the SSN refers to the functional unit hosted by the centroid. *(2.) Transaction Processing* addresses the second challenge. It uses meta data generated by the dynamic SSM in parallel with a time-stamp based ordering of global transactions, as shown in section 3.1. *(3.) Dynamic Topology Handling* concerns the maintenance of the dynamic SSM structure and the disconnected data sources (section 3.2), to address challenge (iii) and (iv).

3.1 TAPCO's Transaction Processing Algorithm

Our system has no assumption on host's mobility. It assumes availability of an on-line thesaurus during the resolution of the transactions. All data local to a host, based on the defined permission, can be modified by other hosts. Each local source preserves local serializability. The global transaction (GT) submitted from the user's mobile host to the system is composed of the data contents and operations (read, write, commit, and abort). A mobile agent, called *GTAgent*, is created for each GT to perform transaction management tasks on behalf of its user. It makes local decisions without user intervention. After the client initiates the GT, it may move or disconnect. When a global transaction is completed, the result is delivered to the user. If the user is disconnected, the result of the transaction will not be lost, but it is kept with the GTAgent and delivered to the user when reconnected. Each SSN and data source maintains an agent, called *NodeManager*, acting as a global transaction manager component (GTM Comp in fig. 1) that interacts with other external entities.

When the user or an application initiates a global transaction, the GTAgent is dispatched to a centroid node. If the cluster-level meta data content does not match the transaction contents, the GTAgent would be forwarded to the upper-level SSN which contains meta data of a broader range of data sources. The GTAgent may travel up or down the SSM according to its semantic content. The GTAgent stops at *Global Transaction Coordinator (GTC)*, which is the lowest SSN that semantically contains related content needed to resolve GT. It acts as the coordinator for a particular GT. The GT is decomposed at its GTC. The resulting global subtransactions (GSTs) are also represented by agents, called *GSTAgent*, which are atomic unit that carry the GST to be executed at a local data source. GSTAgents are dispatched by the GTAgent to the lower SSNs. At each SSN, each GSTAgent is directed to the lower SSN based on the semantic of the GST. Finally, the GSTAgent will arrive at the data source at which its GST will be executed. At anytime, if a GSTAgent realizes that its designated data source is not found, it will notify the GTAgent for global abort.

TAPCO employs a pessimistic approach to resolve conflicts before the actual execution of the transactions to avoid cascading aborts. GSTAgents and NodeManagers cooperate to agree on the serialization order to be used at the local level. Each NodeManager maintains a *Global Order Table*, which keeps the order information of the GSTs that it encounters during the transaction resolution. The order of GSTs in the global order table reflects the global schedule seen by the NodeManager. The order information for each GST includes the timestamp or counter value issued by the NodeManager, and status. Conflicts between GSTs are resolved by TAPCO's ordering rules as follows:

1.) When a GT is resolved at a GTC, all of the GST represented by GSTAgents will have the same timestamp from the GTC upon their creation. When a GST is given the timestamp, an entry is inserted to the *global order table* of the SSN for that GST. Then, the GSTAgent will be given the *global order*, which is the ordered list of all GSTs preceding it in the global order table. The GSTAgent will carry this global order to the next SSN it will visit as directly by the current SSN based on the semantic of the GST.

2.) The NodeManager at the SSN that the GSTAgent visits assigns a timestamp to the GST. When the GSTAgent arrives at an SSN, the information in its global order would be merged to the global order table of that SSN. Thus, the global order information carried by one GSTAgent is transferred to another GSTAgent that later arrives at the same SSN via the SSN's global order table. $GSTAgent_i$ arriving at

the SSN at level *k* before $GSTAgent_j$ would receive lower timestamp than $GSTAgent_j$, and results in the global order $GST_i \rightarrow GST_j$. However, if $GSTAgent_j$ visits the lower SSN (level *k+1*) (which $GSTAgent_i$ must also visit) before $GSTAgent_i$, it will be queued and wait for $GSTAgent_i$ before being assigned a new timestamp and inserted to the global order table to preserve the global order $GST_i \rightarrow GST_j$.

3.) At SSN, the GSTs of the same GT could arrive at different times, but they will have the timestamp of the first GST arriving at the SSN, to allow them to have the same position in the global order seen by the NodeManagers at all involved data sources.

These rules resolve conflicts during the time the GSTAgents propagates from the GTC to the local data sources. Thus, the GSTs that visit the data sources are globally conflict-free if the NodeManager at each local data source guarantee that the global serialization order agreed upon during the transaction resolution is respected. For data sources that apply Timestamp Ordering concurrency control schemes, the NodeManager directly utilizes the existing time order from global level. The NodeManager maintains global order table. When a GSTAgent arrives, the global order carried by the agent is merged into the existing global order table. The NodeManager submits GSTs to the data source according to its global order. For data sources that produce rigorous schedules, or at least recoverable schedules such as strict two-phase locking protocol (S2PL), when the NodeManager receives a prepare-to-commit from a GST, it determines whether the prepare-to-commit operation of that particular transaction will violate the global order. If a GST attempts to enter the prepare-to-commit phase out of the global order determined by the NodeManager, say GST_{wrong}, the NodeManager will delay the GST_{wrong} for a threshold period of time proportional to the number of GSTs that should enter the prepare-to-commit state before the GST_{wrong}. We refer to these GSTs as the predecessors of GST_{wrong}. If all of the predecessors of the GST_{wrong} are able to complete before the threshold period ends, the GST_{wrong} will prepare-to-commit after them; otherwise, it is assumed that the GST_{wrong} indirectly conflicts with its predecessors and should be aborted and restarted. Lastly, if the concurrency control of the local data source is unknown, the forced conflict method is a practical solution [5].

3.2 TAPCO's Dynamic Topology Handling

In this section, we use the scenario in Fig. 3 to explain how our algorithm handles dymanic topology.

3.2.1 Change of Centroid. Nodes participating in the SSM structure are also data sources. These nodes can be either data source only nodes (e.g., node *b* in Fig. 3) and data source/centroid nodes (e.g., node *c*). A centroid is logically SSN. It is periodically informed about the characteristics of other nodes in the cluster to enable the change of centroid when a more appropriate centroid is found. All children are also informed about other children of the same centroid. When a new centroid is selected, the NodeManager at the current centroid and its information including the global order table and GSTAgents in the waiting queue will migrate to the new centroid. In Fig.3, when the NodeManager of SSN000 detects that host *a* should be the new centroid, it informs SSN00 of its migration from host *c* to *a*. After the migration, the NodeManager, now denoted as SSN000', informs SSN0000, SSN0001, and SSN0002 of their new centroid. The NodeManager at SSN00 stays in the same host as the migrating child before the migration (host *c*). Thus, it can no longer stays at host *c* and has choices to move to host *a*, *d*, or *e*. In this case it moves to host *d*, denoted as SSN00'. The change of centroid impacts the SSM and the propagation of the GSTAgents. The GSTAgents determine the next physical host to which it will migrate based on the most recent information at their current centroid. If it fails to contact the NodeManager at the next centroid, it would migrate back to the current centroid to obtain the updated information.

3.2.2 Disconnection of Mobile Nodes. Disconnection-handling scheme is triggered when GSTAgents fail to migrate to the next physical node.

3.2.2.1 Disconnection of Data Source-Only Nodes

As global serialization order is established prior to the execution of the GSTs, the integrity of GTs is not effected by the disconnection of the data sources as long as the agreed on global order is preserved at the disconnected node until reconnected. If the disconnected data source does not reconnect for a time threshold and blocks certain amount of GTs, all the GSTs to the disconnected data source and their corresponding global transaction will be terminated.

3.2.2.2 Disconnection of Data Source/Centroid Nodes

Disconnection of a centroid leads to the selection of a new centroid. In Fig. 3, node *c*, which is the centroid

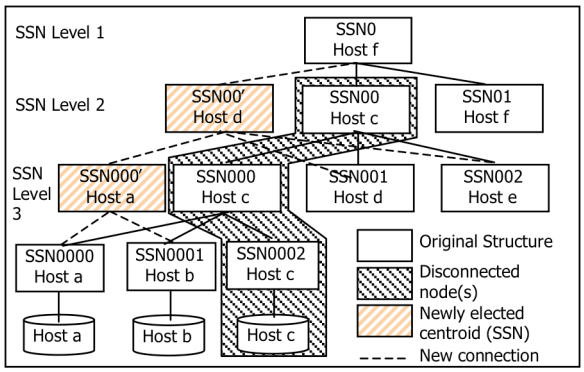

Fig. 3. Disconnection of the Centroid Node or SSN

running SSN00 and SSN000 (dark shaded area), is disconnected. Node *a* and *b* collaborates to elect node *a* as the new centroid, replacing SSN000 (denoted SSN000'). Then, nodes *a*, *d*, and *e*, elect node *d* as the new centroid replacing SSN00 (denoted SSN00'). The global transaction management information for the GSTs designated for data source at node *c* is no longer needed as it is disconnected. Such information for other GSTs, which previously stored at the disconnected SSNs, are restored as follows:

Global Order Table: The global order table contains order information of the already dispatched GSTAgents. Such information can also be found in the global order table of the SSNs immediately below the disconnected SSN. It can be reconstructed at the new SSN by merging the global order table of the SSNs immediately below the old SSN together. For example, global order at SSN0000 and SSN0001 are merged to produce global order for SSN000' at node *a*. We use the creation time of the new global order table as the timestamp of its first entry. Subsequent entries have the insertion time as their timestamp. The global consistency is hold as the global order is preserved.

Waiting Queue: The GSTAgents in the waiting queue are those whose entries are in the global order table at the SSN immediately above the disconnected SSN but not at the SSNs below it. By subtracting these two sets, we can determine the list of such GSTAgents.

GTAgent: The disconnected SSN may be the GTC of some GTs. As the GTAgent cannot contact its GSTAgent during the disconnection, the already dispatched GSTAgents continue to execute and prepare-to-commit. They then contact their GTAgent for global commit and realize the disconnection. Lastly, they would wait for a time threshold before aborting their GST. When reconnected, the GTAgent tries to contact their GSTAgents. If the GSTAgents have terminated, the GTAgent would resubmit the global transaction. If the GST is forced to abort by the data source and the corresponding GSTAgent fails to contact the GTAgent, it will just abort its GST. Other GSTAgents of the same GT will eventually aborted when they realize that the GTAgent is disconnected.

GSTAgent: A new GSTAgent is created to replace the GSTAgent working or waiting at the disconnected SSN. The global order information carried by the disconnected GSTAgent is recovered and passed to its GTAgent, which will create the new GSTAgent.

4. Performance Evaluation

Our system comprises of mobile hosts in ad hoc network moving based on Random Waypoint mobility model. A simulator was developed using NS2 and SimJava. TAPCO is compared against Decentralized Serialization Graph Testing (DSGT). The choice of the DSGT is due to the fact that it is also an agent-based transaction management protocol designed for similar environment. It uses optimistic approach and is tolerant for frequent disconnections. However, DSGT assumes that all transactions are compensatable while TAPCO supports both compensatable and non-compensatable transactions. To explore the full capability of DSGT, all transactions are compensatable in our simulated environment. The environment consists of 10-40 mobile hosts moving at speed up to 10 m/s. Each host may share a data source with probability of 0.8. All hosts submit global transactions to the system. The simulation parameters are given in table 1.

4.1 Simulation Results

An increase in the number of GTs increases the throughput; however, it introduces higher probability of conflict, which could consequently degrade the throughput. Fig. 4 shows the throughput of TAPCO for both global (GT) and local (LT) transactions, and the throughput of DGST for global transactions (GT) (DSGT does not support LT without violating local autonomy). TAPCO always gives higher throughput than DSGT. Its global throughput still keeps rising when the number of GTs reaches 40, traded off by the drop in the local throughput. In contrast, the throughput of DSGT slightly drops as the number of

TABLE 1: GLOBAL & LOCAL SYSTEM PARAMETERS

Global System Parameter	Value
Environment Field Size	400x400m
Maximum Number of Clusters	9
Maximum Number of SSN Levels	4
Maximum Mobility	10 m/s
Pause Time	2s
Probability of Strong Connection	0.5
Probability of Weak Connection	0.5
Transmission Delay during Strong Connection	0.1-0.3s
Transmission Delay during Weak Connection	0.3-3s
Mobile Host's CPU Time	0.005s
Mobile Host's I/O Time	0.01s
Disk Access Time	0.008-0.016s
Time between Checking for Change of Centroid	5-120s
Probability of Change of Centroid	0.2
Number of Data Sources	10
Number of Data Items per Data Source	100
Size of Hotspot (Frequently Accessed Data)	20
Probability of Accessing Hotspot	0.5
Number of GST per Global Transaction	1-10
Number of Operations per Global Subtransaction	1-5
Probability of Read-Only Transaction	0.6
Probability of Read Operation	0.6
Number of Local Application Per Data Source	1
Time between Global Transactions for each Host	0-3s
Time between Local Transactions at Data Source	5-10s

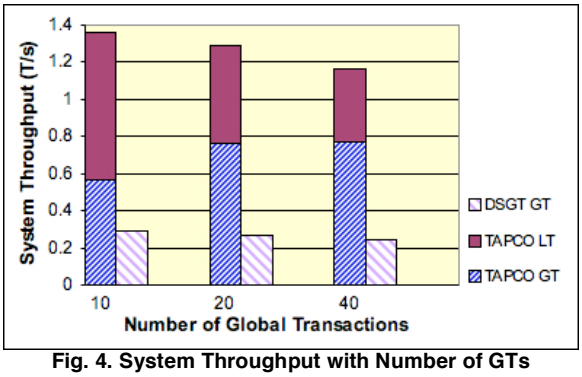

Fig. 4. System Throughput with Number of GTs

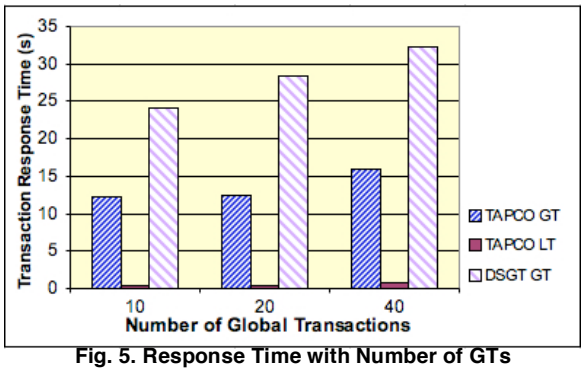

Fig. 5. Response Time with Number of GTs

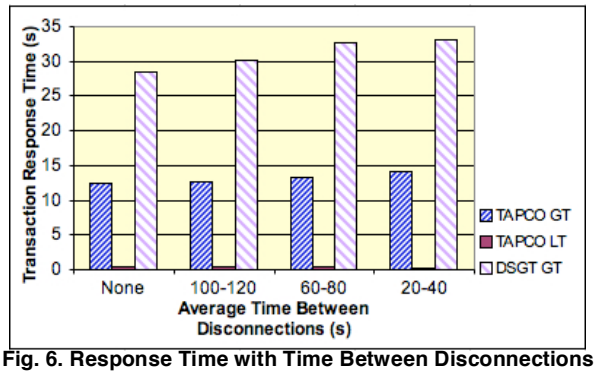

Fig. 6. Response Time with Time Between Disconnections

GTs increases. From Fig. 5, the response time of the successful transactions rises with the increasing number of GTs for both TAPCO and DSGT due to the higher level of conflict. However, the response time of TAPCO is about half of the DSGT due to its parallel processing. The LTs submitted to TAPCO incur ignorable response time because they access a single data source. When the number of GTs are low (low level of conflicts), DSGT has fewer communication messages than TAPCO (not shown). However, this number grows sharply when the level of conflicts increases due to transaction aborts and partial (potentially cascading) rollbacks. In contrast, TAPCO still requires low number of communication messages at high level of conflicts as it avoids cascading aborts.

Fig 6 shows that the response time increases as disconnectivity increases. The response time of TAPCO is always approximately half of DSGT. As the duration of the disconnected data sources increases, the global throughput drops for both TAPCO and DSGT. For TAPCO, as the data sources are still available to the LT, the local throughput slightly increases.

5. Conclusions and Future Directions

We proposed TAPCO, Transactional Agents for Pervasive COmputing, as a transaction management infrastructure for pervasive environment. It utilizes mobile agents to resolve conflicts among transactions and build a globally agreed upon schedule before execution at the local level. TAPCO preserves ACID properties, allows parallel processing of global transactions, and accommodates dynamic changes of ad hoc environment. Unlike some existing solutions, TAPCO does not impose any restrictions on the nature of disconnectivity and the transactions. It handles indirect conflicts without violating the local autonomy of the data sources. TAPCO can be improved by integrating intelligence to the mobile agents to enable them to learn, initiate and process the transactions, and negotiate with other agents or data sources.

10. References

[1] Brayner A., Alencar F., A semantic-serializability based fully-distributed concurrency control mechanism for mobile multi-database systems, Proceedings 16th International Workshop on Database and Expert Systems Applications, 2005.

[2] Brayner A., Filho J., Sharing Mobile Databases in Dynamically Configurable Environments, LNCS Vol. 2681/2003

[3] Bright M. W., Hurson A. R., Pakzad S. H., Automated Resolution of Semantic Heterogeneity in Multidatabases, ACM Transactions on Database Systems, Vol 19, 1994, pp: 212-253.

[4] Dirckze R. A., Gruenwald L., A pre-serialization transaction management technique for mobile multidatabases, Mobile Networks and Applications Vol 5 (4), Dec 2000, pp: 311–321.

[5] Georgakopoulos D., Rusinkiewicz M., Sheth A., On Serializability of Multidatabase Transactions Through Forced Local Conflicts, In Proceedings 7th IEEE International Conference on Data Engineering, April 1991, pp: 314 – 323.

[6] Haller K., Schudt H., Turker C., Decentralized Coordination of Transactional Processes in Peer-to-Peer Environments, ACM Conference on Information and Knowledge Management, 2005

[7] Jiao Y., Hurson A. R., Application of mobile agents in mobile data access systems – a prototype. Journal of Database Management, Vol.15 (4), 2004, pp: 1-24.

[8] Lim J. B., Hurson A. R., Transaction processing in mobile, heterogeneous database systems, IEEE Transactions on Knowledge and Data Engineering, 2002, pp: 1330-1346.

[9] Perich F., Joshi A., Finin T., Yesha Y., On Data Management in Pervasive Computing Environments, IEEE Transactions on Knowledge and Data Engineering, Vol.16, 2004, pp: 621-634.

[10] Perich F., Joshi A., Yesha Y., Finin T., Neighborhood-Consistent Transaction Management for Pervasive Computing Environments, Database and Expert Systems Applications, LNCS Vol. 2736/2003

[11] Yang B., Hurson A. R., Semantic-Aware and QoS-Aware Image Caching in Ad Hoc Networks, IEEE Transactions on Knowledge and Data Engineering, Vol. 19 (12), 2007, pp: 1694-1707