Data Management in Mobile Environments

Katsaros Constantinos
MSc in Computer Science
Athens University of Economics and Business

1 Introduction

It is commonplace that the rapid advances in wireless communications, electronics and networking have resulted in a new computing paradigm, *mobile* or *nomadic computing* [10]. An increasing number of new autonomous, portable devices (Pocket PCs, Palmtops, PDAs, mobile phones) has become a significant part of every day life and work, leading to a decentralized, location independent, wireless computing environment. The extended computing capabilities of such devises have made them suitable for providing information services complementing traditional stationary hosts. Personal mobile devices may now store and process data, giving the opportunity to develop novel applications.

However, unlike traditional wired communications, the deployment of these new services faces several important restrictions due to the wireless-mobile environment. Low bandwidth and battery life as well as frequent disconnections, due to energy saving measures and changes of location, are not negligible factors since they put constraints on the availability and integrity of the offered service. Furthermore, in a highly dynamic environment - in terms of mobility and changes of location - a location management mechanism is necessary in order to assure that mobile hosts can be located and reached at any time [10].

In a data-centric point of view, the restrictions mentioned above introduce several issues that need consideration. The very nature of a mobile environment indicates the *distribution* and *heterogeneity* of data. Either following a completely symmetric peer-to-peer architecture or a client-server one, mobile nodes must be able to reach the desired data in a cost efficient way. Data transfers must be minimized so that wireless environment's limitations do not result in a degraded service and mechanisms must be deployed in order to confront the frequent disconnections and achieve high data access performance, data availability and consistency. Several approaches have been proposed in this direction, including *replication*, *caching*, *semantic caching* and *partial answers*, which are discussed in the remainder of this document. Furthermore, other approaches (some of them *service-oriented* and *context-aware*) have been proposed in order to solve the problem of the heterogeneity of data. These are discussed in the last section of this document.

2 Data Access

Considering the above-mentioned limitations of a mobile environment, a critical issue that arises is the obtainability of data. Major approaches on this issue are discussed in the following.

2.1 Data Broadcasting

Data broadcasting (data push) is regarded as the main method of information dissemination in wireless networks [21]. A lot of research has been done in this field aiming at the improvement of the database server's responsiveness and client data availability [17,18,19,20]. All of these schemes try to reduce the energy consumed on the client by efficiently organizing the contents of the broadcast (based on popularity and/or correlation of data items). But in a mobile server context (for example mobile ad-hoc networks) these techniques fail to address the limited energy and bandwidth problem since they assume that a server has unlimited power supply and may even continuously be active processing and broadcasting. Obviously, traditional mobile broadcast methods fail to deal with server mobility and power limitations [7]. Apparently, similar issues arise as in the case of broadcast based caching invalidation as discussed in the next section. It is not energy efficient that all mobile servers broadcast the same data or some server broadcasts the same data multiple times in order to serve similar requests.

In this direction, two algorithms were proposed in [21] called *adaptive broadcasting* and *popularity based adaptive broadcasting* which address the issues of client and server energy limitation and timing constraints on requests in an mobile ad-hoc network (MANET).

In the first algorithm, a number of mobile servers and clients co-existing in the same area is assumed. Full data replication among the servers is also assumed, posing though a question about the synchronization-updating mechanism. The data in a server's local database is divided into two groups: frequently requested data, called hot data, and less frequently requested data, called cold data. This is accomplished by maintaining a *request frequency* (RF) factor for each data item. Each server periodically broadcasts its power level and location information so that in the end of a period, all servers know this information about all other servers. The server with the highest power is considered to be the leader. It is responsible to schedule the data broadcast of other servers. This means that it partitions hot data into portions. The higher the power level of a server, the bigger portion of the data it will be assigned to broadcast. The lower the power level of a server, the hotter the data will be. This is so because a server can, additionally to the broadcast, reply to data on-demand requests (data pull). Thus, a server transmitting the hottest data is less probable to receive explicit requests, saving thus some energy that has already reached low levels. The adaptation characteristic of the algorithm lays on the fact that the contents of a broadcast dynamically change following the changes in the RF factors. As a data item becomes more popular, it gets included in the next broadcast either by replacing a less frequently requested item or by being appended to the broadcast content.
This algorithm suffers from the following drawbacks. First, there is a large communication overhead when updating the RFs, second there are no request deadlines and client movements are not considered in the calculation of the RFs, third

there is a single point of failure (the leading server), and forth in sight of a new request some older ones may starve.

The second algorithm was proposed in order to address the above-mentioned drawbacks. In this algorithm *popularity factor* (PF) is used. This factor mirrors the importance of a data item. Whenever a client posts a request for a data item, the corresponding PF increases. If the time elapsed since the request of this item exceeds a predefined threshold (*residence latency*, RL) the PF factor decreases. If a server has not received any requests for a long time it will switch to doze mode.

In conclusion, several crucial issues must be considered when talking about data communication in a wireless-mobile environment. First, power consumption must be minimized and this imposes several restrictions on the data communication scheme. Multiple identical data broadcasts and data requests have to be eliminated. A carefully designed broadcast scheme can minimize the power consumption due to explicit data item requests but on the other hand we do not want any redundant transmissions. It must be kept in mind that servers may also be nomadic. Obviously, there is a tradeoff between scheduled broadcasts and on-demand requests. Second, time restrictions must be posed on any data communication method. The consumption of energy is maximal when a mobile node is in active state. This means that a mobile client cannot afford waiting a long time for a data item to be broadcast, not only in terms of power consumption but also in terms of system performance (response time). Therefore, there is need either for some strict, but not restrictive, time scheduling [5,6] or an indexing technique as proposed in [22]. In addition to that, special care must be taken in case of multiple mobile servers in the same area. Simultaneous broadcasts will result in collision wasting power and time both on the servers and the listening clients. A leading server can carry out the coordination of broadcasts as proposed in [21] (with the above mentioned single-point failure drawback). Third, as in the case of broadcast based caching mechanisms, special care must be taken for the consistency and integrity of the data to be ensured. Full replication methods as in [21] impose severe restrictions on this goal while partial replication seems to be a rather difficult task as discussed above. When portions of the network become separated for a time, keeping data accurate may become impossible [7]. Fourth, the existence of peer-to-peer communication mechanisms could improve the performance of a data communication scheme. Near-by mobile clients could share data without the interference of a server entity. This issue is covered in the remainder of this document.

2.2 Replication

Replication refers to the action of creating local or nearly located copies of useful data items in order to avoid later communication overhead or even unavailability in case of on-demand retrieval from their original location. The need for data replication derives directly from the restrictions in a mobile/wireless environment. Disconnected or poorly disconnected nomadic users rely primarily on local copies of the required data in order to achieve availability and reliability. However, these restrictions also apply in the case of these replication mechanisms' deployment. Full replication results in additional energy consumption and heavy traffic generation [7]. Consequently, traditional, full replication techniques designed for static infrastructure seem to be inappropriate for a mobile environment [15]. On the other hand, *partial replication*

can be regarded as an alternative solution but it also hides some important issues that need deep consideration. Partial replication may be achieved by partitioning the data (for instance a database). In the mobile context and especially in peer-to-peer systems this task could prove to be very difficult as the data placement problem in such systems is *NP-complete* [8]. Moreover, updating partial and/or full replicas is considered as a difficult problem. In general, data items with low update rate should be replicated in the environment of a client that wishes frequent access to them. In contrast, if a data item is updated more frequently than a user wishes to access it, then it is probably more sufficient for the client (and the server) to access it on-demand [23]. However, this problem is similar to cache updating discussed below, in that in both cases the objective is to update stale data on a mobile host. The difference between these situations is the size of the data that need update. Usually replicas are large amount of data in contrast to caches. Several approaches have been proposed for the cache (in-) validation problem, as described in the following section.

In conclusion, several issues have to be addressed in order to clearly determine the potentials of data replication in mobile environments. First, there is the need for any-*to-any* communication [15]. A client-server approach may not prove scalable. Finding stable nodes to host replicated data may prove hard in the context of a highly dynamic network topology. If the number of mobile users increases, then this approach may no longer be viable since heavy traffic is caused in the network due to the redistribution of the data in cases of node movement. It is obviously preferable that mobile users communicate directly with each other and get the desired data replicas from near-by users [1]. Of course, this requirement entails an efficient knowledge partitioning mechanism as described above, as well as a scalable updating mechanism. Second, there is need for detailed replication control. Unnecessary data transfers must be avoided in order to save disc space and bandwidth. This means that individual object selection must be possible. Obviously, transferring a large-granularity container will result in transferring unneeded objects [15]. Thus, the replication mechanism must be flexible. Third, replication is often based on *a priori* decisions of the user. The user is often expected to explicitly state their intensions about moving to another location. This characteristic may be restrictive for the way a mobile user moves from place to place and/or hindering for the systems adaptation to a possible new location. Mobile users cannot always predict their location. Fourth, in an environment with autonomous data sources, replication may not be possible simply because the data source forbids it [3].

It seems from the above, that replication in a highly dynamic mobile environment is a difficult task. This is the reason that attention is drawn to flexible caching schemes (see next section). However, as long as efficient solutions are provided for the problems discussed above, replication could increase the systems data availability and data access performance.

3 Caching

Caching is a widely used mechanism for improving data access performance and availability. The main difference between caching and replication is that the former occurs after the retrieval and use of the data while the latter in an *a priori* way. Especially in a wireless, mobile environment, caching of frequently accessed data in a

mobile node's local storage can reduce energy and bandwidth consumption as well as query delays, while at the same time increasing the system's flexibility in cases of disconnection. However, a fundamental issue when considering caching policies is data consistency. A client must always ensure that data in its cache is up to date in order to be able to provide valid responses in submitted queries.

3.1 Broadcast Based Caching Schemes

Various caching schemes have been proposed, addressing the problem of cached data consistency. In these schemes, a client-server architecture is considered, in which the original data reside on a stationary host (server) that serves client requests. The server broadcasts special messages (invalidation reports) in order to help clients keep their cached data consistent.

Two main invalidation strategies exist, depending on whether the server maintains the state of each client's cache [6,16]. In the *stateful* server strategy, the server maintains information about the data items each client keeps in its cache. Thus, it is responsible to inform each client about any updates on the data items they have cached. To do so, invalidation messages are sent to each client holding an old copy of an updated data item. This strategy is not efficient when clients are frequently disconnected, as they cannot be informed about the invalidation of their cache. This results in the loss of their cache in view of the use of stale data [16]. Moreover, in case a mobile client moves, it must de-register from the server and register to another. This is obviously a situation where mobility is hindered. Last but not least, a client gets informed about the change in the state of a data item even if it does not plan to use it. This is a potential waste of bandwidth [6]. An example of this approach is the Andrew File System [24]. In the *stateless* server strategy, the server maintains no information about its clients' caches. Mobile clients are responsible for updating their cache. To do so, they communicate with the server in order to learn about any possible updates. This approach is obviously power and bandwidth consuming. An example of this approach is the Network File System [25].

Another approach on the subject is the mechanism of *timestamps* (TS) [6]. In this cache invalidation method, the stateless server broadcasts invalidation reports periodically, every L seconds. These reports consist of the *id* of all data items that changed during the last w seconds (where $w \geq L$) coupled with the corresponding *timestamp*, which declares the time that this item last changed. Upon reception of the invalidation report, mobile clients check their cached data. If they find an outdated item (the new timestamp in the invalidation report is this case larger than that of the cached item) they drop it from the cache. If an item is not reported in the invalidation message then its timestamp in the cache is updated to the timestamp of the invalidation report. If the mobile client realizes that the interval between to successive invalidation report receptions is bigger than w, then the whole cache in dropped. A mobile client may produce an answer to a query only when it has received the next invalidation report, which means only after it has checked the consistency of its cache.
A variation of the TS method is that of *Amnesiac Terminals* (AT). In this method the server periodically informs the clients about the items changed since the last invalidation report. If a mobile client gets disconnected for a period of time, it has to rebuild its cache from scratch.

The evaluation of these schemes shows that the TS method is more suitable for mobile clients that is less probable to be disconnected (*workaholics*) while the AT method is more suitable for mobile clients that are in a disconnected state most of the time.

The cache invalidation method described above has two major drawbacks which were faced in the *low-latency cache invalidation method (UIR)*[16]. First, in the TS method a client may answer a query only after it has received the next invalidation report. This means that a client must wait *L/2* seconds in average, in order to validate its cache and answer a query. The proposed solution to this delaying factor is the introduction of short intermediate invalidation reports (*updated invalidation reports, UIR*). Between the broadcasting of two consecutive invalidation reports, (*m-1)* UIRs are broadcast to the clients every *1/m* seconds. These reports refer only to the data items that changed since the last invalidation report (similarly to the AT method). Thus, a client may be able to answer a query only after *1/m* seconds after the reception of the query. Second, in the TS method, if an updated item resides on multiple clients' caches, then separate queries will be issued and answered for these caches to be updated. This is obviously a bandwidth-consuming situation. The proposed solution here is to keep track of the requested items. When the server receives a request for a data item it refrains from responding immediately. Instead it saves the id of the requested data item in a list L_{bcast}. This list is broadcast immediately after the next invalidation report. All clients listening to this report will also learn about the data items that are going to be transmitted by the server. The server broadcasts these items after the broadcasting of L_{bcast}. All clients interested in the broadcast data item will have already been informed about its transmission and will simply save it without issuing any data item request. However, all clients must wait for the transmission of L_{bcast} before queries can be answered, resulting in longer delays [5]. The same problem is addressed in [5] where the *Delayed Requests Scheme (DRS)* is proposed. This method aims at reducing the uplink traffic due to identical data item requests. In this scheme, clients with invalidated data items refrain from posting a request to the server for a period of t_{wait} seconds. During this period they listen to the downlink channel to see if the server is already planning to broadcast these items. In order for a client to learn so, the server, upon reception of a request, broadcasts a notice message informing the clients of its intention to broadcast the requested items. If eventually the client does not get informed about the scheduled broadcast of the data item, it posts a request for it.

Neither of the above methods addresses the problem of potentially very large invalidation reports. Both in TS and UIR methods, the size of an invalidation report is proportional to the number of updated data items. Thus, in cases of high update rates, the invalidation reports become large, forcing mobile clients to spend their limited bandwidth and energy resources. The *validation-invalidation reports* scheme (VIR) [5] aims at solving this problem. The idea is that when the number of updated data items increases (in general: more than the half the data items have to be invalidated), the number of unchanged – and still valid – data items shrinks. Thus, it is more efficient to send *validation reports* in this case, instead of the larger invalidation ones. These reports consist of the id of each validated data item (*id*) and the corresponding timestamp (*t*) that indicates the time of the last change of the item. Obviously, the following condition must hold: $t \leq T_{report} - L$.

A common advantage of all the above approaches lays on the fact that broadcasting is a cost effective way of communicating cache validation information to all mobile clients [5], since the cost of broadcast communication in a wireless environment is independent of the number of recipients. On the other hand, a particular constraint posed by this approach is that mobile clients have to be listening for these messages. Obviously this is an energy-consuming situation, which however has been dealt with [6].

It must be pointed out that all the above approaches were studied in the context of stationary servers and mobile clients. In a different architecture, where mobile nodes hold databases, the above techniques for creating the cache and preserving its consistency, cannot be applied straightforwardly. Power and bandwidth restrictions make it costly for a mobile node to periodically broadcast invalidation reports.

3.2 Semantic Caching

Although it has been proved that traditional broadcast based caching schemes improve data access performance and data availability, it is difficult for a client to determine if a query could be answered entirely based on locally cached data, forcing it to contact the database server [4]. This holds because of the lack of *semantics* in these caching mechanisms.

An additional caching mechanism is proposed [4] in order to solve this problem. In *semantic caching*, data is cached as a collection of possibly related blocks. The relation of these blocks comes from the fact that they represent the results of previously evaluated queries. In other words, each caching unit accompanied with its semantic description can be regarded as a materialized view. In this way, a client may compare a new query with the description of these views (called *restrict condition*) and decide whether it can produce an answer or it has to post a data request to the server. More specifically, when a query is submitted to the clients, a query handler module analyzes it to determine whether it can be answered locally. If so, the query is executed on the cached data without any communication establishment with the server. In this case the query is said to be *completely self-answerable* [4]. In case the query cannot be answered completely using the cached data (*partially self-answerable*) then it is divided into two parts, a *probe query* that will be answered using the cached data and a *supplementary query* that describes the missing data that need to be transferred from the server. The whole process is called *query transformation*. The results of these two queries are integrated to form the original query's result. Each query result is stored in the cache as a *cache fragment* for future use. These fragments can be broken down into smaller, finer ones (called *sub-cache fragments*), in order to achieve better caching granularity (*cache fragmentation*). If the client is disconnected during the above process only the result for the probe query will be returned to the user.

The updating mechanism of a semantic cache is based on the lazy, data pull method. Each caching unit is associated with a timer, whose expiration triggers the client to post a request to the server for the updated data.

Another issue about semantic cache management is the materialization of the views a semantic cache contains. As mentioned above, each caching unit can be regarded as a

materialized view on the original data. However, multiple views have overlapping areas (over the same data) raising the point of selection of the views to be materialized. A greedy approach would be to materialize all views to provide optimal performance in query processing. This approach would result in extensive storage requirements, which are difficult to satisfy in a mobile device. An alternative to this is to materialize the most frequently accessed views. This is done during cache fragmentation process.

This type of caching yields several important advantages. Partial or complete answers can be retrieved from a semantic cache reducing the power and bandwidth needs. In the case of a complete answer it is more than obvious that semantic caching can save significant resources. Moreover, this characteristic can prove very useful in cases of disconnection. These cases are frequent in a wireless environment and this method can significantly improve the reliability and the availability of the system. Moreover, semantic caching offers a great degree of flexibility in the cache management since (as mentioned above) it can support various degrees of caching granularity. In fact, with semantic caching, the granularity of cached data is the result of the query [27]. In consequence, semantic caching can result in reduced space requirements, in addition to its network traffic efficiency. In cases where the projection operator is supported by the semantic caching scheme [4], the above advantages are amplified since the projection operator is very useful at pruning unnecessary attributes.

3.3 Cooperative Caching

Cooperative caching is the caching scheme that allows the sharing and coordination of cached data among multiple nodes [1]. This scheme can be used in ad-hoc networks in order to reduce query delay and message complexity as well as power and bandwidth consumption. It can also be considered as an efficient alternative to replication schemes in environments with frequent topology changes, as described above.

In an ad-hoc network, each node may serve neighboring nodes as a forwarder-router or even as a gateway in cases where a neighboring node cannot establish communication with any other node. A set of cooperative caching schemes is proposed in [1] for such an environment.

In the first scheme, C*ache Data*, a node monitors passing-by data items. If it finds out that there are many requests for a data item or there is enough free cache space, it caches it. To avoid situations where all nodes in the path between the data source and the requesting node cache a frequently requested data item, a node does not cache a data item if all requests come from the same node. In this case, it is more efficient to store the item closer to the requestors (the closer common node in the paths from the requestors to the source) and avoid thus all unnecessary traffic along the path as well as storing data in other nodes. Using this rule, at least the requesting node will cache the data item.

In the second scheme, *Cache Path*, a node caches the destination of passing-by data items in order to keep track of an alternative location of various data items (apart from that of the data source). Based on the underlying routing mechanisms, a node can then

forward incoming requests to the closest of the known holders of the requested item. To reduce network traffic, a node caches an item's destination only when it is very close to it. This ensures that caching information will be useful since a very distant destination will never be preferred for forwarding a request to.

It must be noted, that the above mechanisms require the constant operation of the mobile nodes, since processing of all passing-by data items is needed. This is obviously power consuming. In addition to that, heavy network traffic is generated due to the frequent employment of the underlying routing algorithms for the distance calculations to take place.

4 Partial Answers

Another method proposed for increasing a systems performance under restricting conditions, such as data source unavailability, is *partial answers* [3]. In this approach, the main target is to provide the system with the ability to return answers to queries even in cases where the sources of the required data are not accessible. Such a condition is obviously very often in a mobile environment where clients suffer from frequent disconnections. The method for extracting partial answers is suitable for systems where data is distributed among several sources.

The main idea here is that in cases where not all data sources involved in query are available, we can still take advantage of the ones that are available and provide some answer to the user. When a query - involving multiple data sources- is submitted by the user, it is evaluated by the system. If all data sources are available then the system returns the complete answer to the query. If no data sources are available then the answer is null. If some of the data sources are not available during the query evaluation then a partial answer is returned. This answer consists of the data accessed from the available data sources, a query on the unavailable data sources and some additional information gathered during evaluation, e.g. the list of data sources that were available or the ones that were not.

Partial answers can be either *transparent* or *opaque*. In the first case, we can take advantage of the data returned in the partial answer in order to extract a part of the complete information. This can be accomplished through the use of a *parachute query*. A query of this type targets at the data obtained from the available data sources. Multiple parachute queries can be submitted. A problem that turns up here is that the effectiveness of a parachute query depends on the structure of the intermediate results during execution of the original query [3], which is determined by the query optimizer. This structure may obstruct the extraction of useful information because query optimization is not targeted at a particular parachute query. One solution is *constrained optimization*, where the parachute query is submitted together with the original one so that the query optimizer may produce an execution plan suitable for both the queries. In this case, the possibility that a certain data source may be unavailable must be taken into account. Another option is *unconstrained optimization*, where the optimization of the original query is done regardless of the parachute query's structure. In this case, query optimization is simpler but the effectiveness of parachute queries is reduced. To improve on this, intermediate results can be retained during the evaluation of the original query. Returning to the case of

opaque partial answers, the query returned inside the partial answer can be later re-submitted in order to obtain the missing information. The whole process can be repeated several times until we have a complete answer for the original query. Thus, we are able to extract all the required information even if all data sources are not available at the same time. It suffices that all data were available at least once.

Another important feature of this approach is the embedment of the *query scrambling* functionality in the query evaluation process. Query scrambling is used to handle the problem of delayed responses from the data sources. When a data source delays its response during the evaluation of a query, the data source is marked as unavailable and skipped without blocking the system. Thus, the system has the opportunity to proceed with the rest of its tasks letting the partial answering mechanism solve the problem of this data source's unavailability.

5 Heterogeneity and Service-oriented approaches

Another important issue regarding data management in a mobile environment is data heterogeneity. In a nomadic computing paradigm, it would be naïve to assume that a uniform, universal representation of data for all computing nodes surely exists. We cannot rely on the existence of a predefined, global schema [2]. In addition to that, a large variety of mobile devices is expected to further enhance this diversity. Thus, it is important to provide this new computing environment with the necessary mechanisms so that information dissemination is feasible, overcoming existing diversities. In addition to that, mobility introduces one more factor, *context-awareness* [11,12,13,14]. Data may be location dependent [9,10] and this must be seriously taken into account when considering data management. Context-awareness may prove helpful in providing mobile nodes with the desired information, as it may help specifying the data required by a user [11,12,13,14].

A *data-centric* view of such an environment relies on the use of *meta-data* [26]. Meta-data is used to facilitate data discovery as well as us to speed up data processing. In other words, meta-data can be used by a mobile node to determine mappings between its schema and its neighbors' ones. In order to achieve that, metadata primarily describe *content data*. Content data is the actual data a mobile node carries. In some cases [11,12,13,14] content data may also contain location dependent information (for example, current location and temperature). Furthermore, metadata can be used first, to ensure suitable data representation for the various types of mobile devices, second, to provide personalized services and third, to optimize routing. In these cases, metadata would be used to describe the type of the user's device, the user's preferences and the user's mobility patterns [10] (*profile data* [26]). There are several widely accepted languages and tools appropriate for the deployment of meta-data such as XML and RDF.

An approach on the use of meta-data is given in [2]. In this peer-to-peer based, architecture, meta-data are maintained for each relationship, name and attributes. These metadata contain keyword/descriptions of the items they represent, which are provided by the user upon the creation of the item. All these elements are stored in a *Local Dictionary*, on the mobile node. Meta-data of sharable items are also associated with the *Export Dictionary*. When a mobile node wishes to find data relative to a submitted query, it launches *DBAgents* modules that are responsible to find potential

suitable relations in the neighbors' Export Dictionaries. This is done through keyword searching. The matching relations (meta-data, database name and location) are returned to the querying node. This is done for two reasons. First, to provide the user with the necessary information for him/her to chose the most suitable relation, if any. The system lets the user make this choice in order to avoid situations where data may be syntactically the same (having the same keywords) but semantically different. In these cases, significant resources (power and bandwidth) are wasted with no avail. Second, the metadata of the user's choice can be used for future search process.

A different architecture is used in [11,12,13,14]. *DBGlobe* (elsewhere called *MobiShare* [13]) uses a two-layer architecture. In the first layer, individual peers, called *Primary Mobile Objects* (PMOs), communicate directly with each other in a peer-to-peer mode. In the second layer, administration units, called *Administrator Servers* (ASs) or *Community Administration Servers* (CASs), are employed to coordinate the systems operation inside a certain area analogous to a cell, in terms of cellular telephony. In DBGlobe a service-oriented approach is employed, in that data are encapsulated in services. A service is accessible through messages, language and platform independent, and produces output messages (results), which are machine-oriented (with the use of profile data as discussed above). Every PMO registers with the local CAS by providing metadata information similar to that described earlier, giving the CAS the necessary information to build a service directory. This directory lists all services offered by PMOs in the cell. The CAS uses a service ontology (taxonomy [13]) with a hierarchical structure in which semantically similar services are related to the same node in the tree, e.g. to the topic research [12]. When a PMO is searching for specific data, it firsts contacts its CAS. The CAS traverses the service ontology to locate the best matching service for the submitted request (usually a string complemented by lexicographical matches using a thesaurus) and sends back the results. The user chooses the desired service and accesses it directly.

6 Conclusions

The main targets for a mobile data management system is to ensure data availability and consistency even in cases of disconnection, which are frequent in these systems. The problem is that, in a mobile computing environment, energy and bandwidth restrictions affect all data management related issues making the achievement of these goals a difficult task. What derives from the above is that further research needs to be done, as most of the solutions proposed today assume stationary data sources. In view of the emergence of numerous data producing devices [9] a lot of issues described above must be addressed.

References

[1] Y. Yin and G. Cao, "Supporting Cooperative Caching in Ad Hoc Networks," *IEEE INFOCOM,* March 2004.

[2] B.C. Ooi, Y. Shu and K. Tan, "Relational Data Sharing in Peer-based Data Management Systems," *Proceedings of theACM SIGMOD Conference,* September 2003.

[3] P. Bonnet and A. Tomasic, "Partial Answers for Unavailable Data Sources," Proceedings *of the Conference on Flexible Query Answering Systems, Roskilde, Denmark,* 1998.
[4] K. C.K. Lee and H.V. Leong, "Semantic Query Caching in a Mobile Environment," *Mobile Computing and Communications Review*, *Volume 3, Number 2,* April 1999.
[5] K.Y. Lai, Z. Tari and P. Bertok, "Cost Efficient Broadcast Based Cache Invalidation for Mobile Environments," *ACM,* March 2003.
[6] D. Barbara and T. Imielinski, "Sleepers and Workaholics: Caching Strategies in Mobile Environments," *VLBD Journal,4 ,* March 1995.
[7] L.D. Fife, "Research Issues for Data Communication in Mobile Ad-Hoc Network Database Systems," *Proceedings of theACM SIGMOD Conference,* June 2003.
[8] S. Gribble, A.Halevy, Z. Ives, M. Rodrig and D. Suciu, "What can Databases Do for Peer-to-Peer?" *Proceedings of the Workshop on the Web and Databases (WebDB)*, 2001.
[9] T. Imielinski and B.R. Badrinath, "Wireless Graffiti – Data, data everywhere," *Proceedings of the 28th VLBD Conference,* 2002.
[10] T. Imielinski and B.R. Badrinath, "Querying in highly mobile distributed environments," *Proceedings of the 18th VLBD Conference,* 1992.
[11] A. Karakasidis and E. Pitoura, "DBGlobe: A Data-Centric Approach to Global Computing," *Proceedings of the 22nd International Conference on Distributed Computing Systems Workshops,* 2002.
[12] E. Pitoura, S. Abiteboul, D. Pfoser, G. Samaras and M. Vazirgiannis, "DBGlobe: A Service-Oriented P2P System for Global Computing," *ACM SIGMOD Record,* September 2003.
[13] E. Valavanis, C. Ververidis, M. Vazirgiannis, G. C. Polyzos and K. Norvag, "MobiShare: Sharing Context-Dependent Data & Services from Mobile Sources," *IEEE/WIC International Conference on Web Intelligence (WI'03),* October 2003.
[14] C. Ververidis, S. Valavanis, M. Vazirgiannis and G. C. Polyzos, "An Architecture for Sharing, Discovering and Accessing Mobile Data and Services: Location and Mobility Issues," *In proceedings of the LOBSTER Workshop, Mykonos,* 2002.
[15] D. Ratner, P. Reiher, G. J. Popek and G. H. Kuenning, "Replication Requirements in Mobile Environments," *Mobile Networks and Applications, Vol.6, Issue 6,* November 2001.
[16] G. Cao, " A Scalable Low-Latency Cache Invalidation Strategy for Mobile Environments," *ACM*, 2000.
[17] D. Aksoy and M. Franklin, "Scheduling for Large-Scale On-Demand Data Broadcasting," *Proceedings of the 12thInternational Conference on Information Networking,* January 1998.
[18] V. Grassi, " Prefetching Policies for Energy Saving and Latency Reduction in a Wireless Broadcast Delivery System," *Proceedings of the 3rd ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems,* 2000.
[19] Y. Guo, M. Pinotti and S. Das, "A New Hybrid Broadcast Scheduling Algorithm for Asymmetric Communication Systems," *ACM Mobile Computing and Communications Review,* 2001.
[20] E. Yajima, T. Hara, M. Tsukamoto and S. Nishio, "Scheduling and Caching Strategies for Broadcasting Correlated Data," *Proceedings of the 16th ACM Symposium on Applied Computing,* March 2001.

[21] L. Gruenwald, M. Javed and M. Gu, "Energy-Efficient Data Broadcasting in Mobile Ad-Hoc Networks," *Proceedings of the International Database Engineering and Applications Symposium,* July 2002.
[22] I. Imielinski, S. Viswanathan and B.R. Badrinath, "Energy Efficient Indexing on Air," *Proceedings of the ACM SIGMOD Conference,* May 1994.
[23] Y. Huang, P. Sistla, and O. Wolfson, "Data Replication for Mobile Computers," *Proceedings of the ACM SIGMOD Conference,* May 1994.
[24] K. Kazar, "Synchronization and Caching Issues in the Andrew File System," *USENIX Conference,* 1988.
[25] S. Sandberg, D. Goldberg, S. Kleiman, D. Walsh and B. Lyon, "Design and Implementation of the Sun Network File System," *Proceedings of the USENIX Summer Conference,* June 1985.
[26] D. Pfoser, E. Pitoura, N. Tryfona, "Metadata Modeling in a Global Computing Environment," *19th ACM International Symposium on Advances in Geographical Information Systems (ACM-GIS),* 2002.
[27] M.T. Ozsu and P. Valduriez, "Principles of Distributed Database Systems, Second Edition," *Prentice-Hall,* 1999.