# Dynamic Policies in Selecting a Caching Set for a Distributed Mobile Computing Environment

Ada Fu John C.S. Lui M.H. Wong

June 4, 1995

**Abstract**

The bandwidth of a wireless network is usually very limited as compared to that of a static network. Therefore, caching the right set of data into the limited storage of mobile computers is an important issue in mobile database applications. Caching not only reduces the contention on the narrow bandwidth of a wireless channel, but also improves the response time for read operations. However, we show that the problem of finding an optimal caching set so as to minimize the bandwidth utilization is NP-hard. In order for the system to determine the caching set in real time, two heuristic methods, the baseline algorithm and the greedy algorithm, are proposed. The closed-form solutions of various performance measures for the no caching policy, the baseline algorithm and the greedy algorithm are derived based on the queuing model. Experiments are also conducted for various access patterns. In general, the proposed caching algorithms outperform the no caching policy under all cases. In addition, the capacity of a cell is significantly improved when the caching algorithms are adopted. Our algorithms are simple and efficient, hence they can be readily incorporated into a mobile database system to ensure good response time and the cost-effective usage of the mobile computer memory and the wireless bandwidth.

# 1 Introduction

The potential of mobile computing is recognized by the research community recently. It is foreseen that with mobile computer networks, business men, technical staff, salespersons, and many other kinds of users can make use of the computers and the networks in a more flexible way than ever before. A number of researchers have discussed the related problems and challenges in this area [1, 2, 4, 7, 10, 11, 15]. Challenges of the mobile environment arise because of its unique characteristics which are different from other network systems. For example, some challenges are the mobility of mobile computers, which can frequently connect and disconnect from a stationary host computer [16], the limitation of power supply at the mobile computers [12], and, perhaps the most important of all, the wireless bandwidth limitation and the limited memory capacity of the mobile computers[3, 9].

The concern of bandwidth is much more acute than for stationary computing. From [7], we see that *the cutting edge products for portable wireless communications achieve only 1 Mbps (Megabit per second), 2 Mbps for radio communication and 9-14 kilobits per second for cellular telephony. Whereas for stationary computing, Ethernet achieves 10 Mbps, FDDI reaches 100 Mbps, and asynchronous transfer mode (ATM) can achieve 155 Mbps. Even non-portable wireless networks, such as the Motorola Altair, barely achieve 5.7 Mbps.* On the other hand, the application requirement of functions such as information retrieval, email and file transfer can reach 1 Mbps. Therefore, careful usage of bandwidth is a major concern in the design of protocols for multiple user access to a mobile network. This issue is especially significant for mobile database applications, since a large amount of data is transmitted through the wireless channels.

One way to reduce the wireless traffic is to allow mobile computers to cache some of the frequently used data. When a data is cached, there has to be some form of mechanism to inform the mobile computers of changes to the cached data. This problem is more complicated in a mobile computing environment because of the unrestricted mobility of mobile computers and their frequent connect and disconnect operations. In [3], a number of caching strategies based on periodic broadcasts have been proposed. It is argued that broadcasting updated information is a better strategy than non-broadcasting methods in a mobile environment. However, [3] does not address the problem of how the caching set (set of data to be cached at one or more mobile computers) is selected. Therefore, the message broadcast to the mobile computers contains all the data objects that have been updated in the last period, even though some of the data objects may not be cached in any mobile computer. In [12], two indexing techniques are proposed so that mobile computers can become active only when data of interested is being broadcast. These approaches can reduce the work load of mobile computers, however, it cannot alleviate the contention of the narrow bandwidth.

In [9], the authors deal with the problem of selecting data objects to be cached at a mobile computer in order to minimize the communication between the mobile computer and the stationary host computer. The proposed algorithms are shown to produce solutions that are within a constant factor of the optimal solution. However, the study in [9] is based on a single mobile computer. We find that in the case of a cell with multiple mobile computers, if a broadcasting strategy as proposed in [3] is adopted, then the problem of cost minimization is much more difficult. This is because the communication cost of write operations depends on the caching set of all mobile computers. Hence we cannot decompose the problem of caching for multiple mobile computers into multiple problems of handling each single mobile computer.

In this paper, we consider the situation of multiple mobile computers in a given cell. We find that given the read and write frequencies, the problem of choosing data objects for caching so as to minimize the wireless bandwidth consumption is NP-hard. We then derive two heuristic algorithms, baseline algorithm and greedy algorithm, for selecting the caching set dynamically. The closed-form solutions of various performance measures for the no caching policy, the baseline algorithm and the greedy algorithm are derived based on the queuing model. The performance metrics that we use are the residual bandwidth and the response time of read operations. Experiments are also conducted for various access patterns. In general, the proposed caching algorithms outperform the no caching policy under all cases. In addition, the capacity of a cell is significantly improved when the caching algorithms are adopted. The proposed algorithms are efficient and simple so that the caching set can be determined in real time to adapt to the rapid changes of access pattern. Efficiency is important in our system model, since the number of objects and the number of users are typically large, even a polynomial algorithm of complexity $O(N^3)$ for a database size of $N$ is not quite acceptable. Our algorithms satisfy this constraint in computation complexity and the simplicity of the algorithms would make it very easy for implementation, which is an important factor in system development.

This paper is organized as follows: In the next section we present the system model, describe the problem and give a proof of its NP-hardness. In Section 3 the proposed heuristic algorithms are presented. In Section 4, the closed-form solutions of different performance measures are derived. The results of the experiments are depicted in Section 5 together with a brief analysis. Finally, we conclude our paper in Section 6.

# 2 Statement of Problem

The architecture that we consider is a network of stationary computers (SC) and mobile computers (MC) (see Figure 1). In this network, a number of SC's are connected in a wired distributed system environment. Each SC is responsible for receiving workload generated within its own cell, where a cell is a geographical area. A mobile computer (MC), on the other hand, can traverse from cell to cell and generate workload at will. Since the problem of multiple cells can be seen as multiple problems of each single cell, we shall concentrate on the handling of wireless bandwidth within a single mobile cell. There will be one stationary computer, and multiple mobile computers, and they communicate via the air. The bandwidth of the communication is given by $BW$. Figure 1 illustrates the underline architecture.

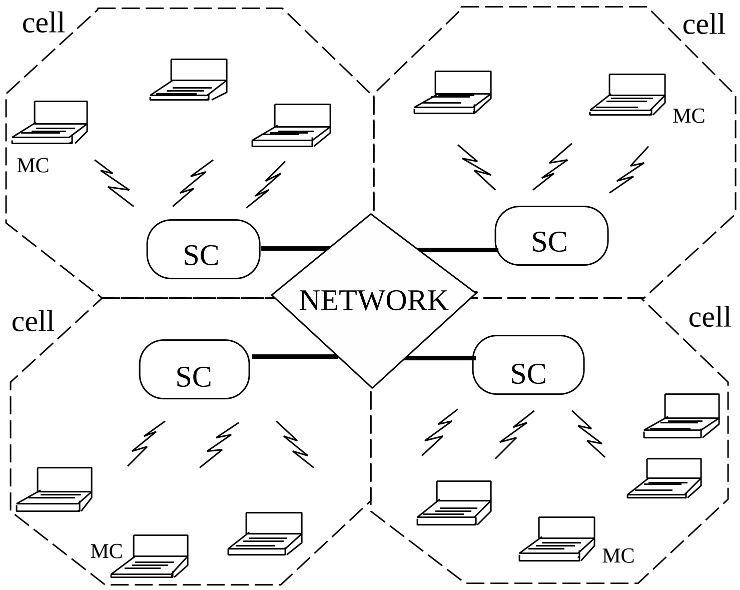



Figure 1: Mobile Computing Environment Architecture

The database consists of fixed size objects that are shared by all users. We assume that the work generated by an MC consists of only read operations. Each MC has a fixed amount of memory (cache) in which copies of data objects are stored. A data object can be updated by write operations generated at the SC and the updated values will be broadcast immediately to the MC's if the object is cached by one or more MC's.

The cost we are concerned with is the communication cost between the MC's and their corresponding SC, or the use of the wireless bandwidth demanded by the read and write operations. If the mobile computers choose not to cache any data, so that the data objects are accessible only from the SC, then each read operation from the MC must be executed via messages to and from the SC within the cell. The key factor which allows us to reduce the wireless bandwidth consumption is that a mobile computer can cache some of its frequently used data objects.

If a data object is cached and its value is "up-to-date", then a read operation from the MC does not need to send any message to the SC in its cell, hence cutting down on the use of the bandwidth. However, once a data object is cached, an update of the object (from a stationary computer) will require some communication to keep the cached value up-to-date. Whereas if there is no cache, then the update of the object does not require this communication. An important point here is that we assume that such an update message is a broadcast from the SC to the MC's in its cells, and the cost of the broadcast does not depend on how many MC's require the message. Hence there is a tradeoff between caching and not caching an object, where the gain or loss depends on the read/write frequencies and read/write message costs.

Let there be $N$ objects in the database, and $M$ MC's in the cell. Our basic protocol operates in terms of time periods of $T$ seconds. At the beginning of each period, the SC broadcasts necessary messages to the MC's. This may include some messages to the MC's that have newly joined the cell. In our algorithms, we need to gather statistical information at each period from the MC's about what data objects they access and how often they do so. This information will help us to determine the Poisson arrival rate for read requests, which in turn will help us to determine what data are to be cached in the following periods. If a data object is not cached by a $MC$, then the read requests from the $MC$ would be known by the SC, since each read request has to go through the SC. However, if a data object is cached by the $MC$, then the read operations will not be known by the SC, since the requests can be satisfied by the local copy in the $MC$. Hence some information gathering will be necessary for such cases, we assume this will be done by some communication generated at the beginning of each period.

Within each time period, a SC in a mobile cell may issue a write request on any object, and if the object has been cached in one or more MC's in the mobile cell, then the new value will be broadcast by the SC to all MC's in the cell. Note that the SC in this cell may not be the originator of the write request. However, an update in one cell can be propagated to another cell via the wired network. Hence we may consider the write frequency as the the total write frequency to the objects in the entire network. Figure 2 illustrates the basic broadcast protocol.

The following lists some symbols used in this paper:

$T$ = time period of collecting statistics.
$B_i$ = Buffer size at mobile computer $MC_i$.
$\Gamma_{ij}$ = 1 if object $j$ is cached in MC $i$, 0 otherwise.
$\lambda^r_{ij}$ = Poisson arrival rate for read request generated from MC $i$ to object $j$.
$\lambda^w_j$ = Poisson arrival rate for total write request to object $j$.
$C_r$ = read cost for reading a unit of object not cached.

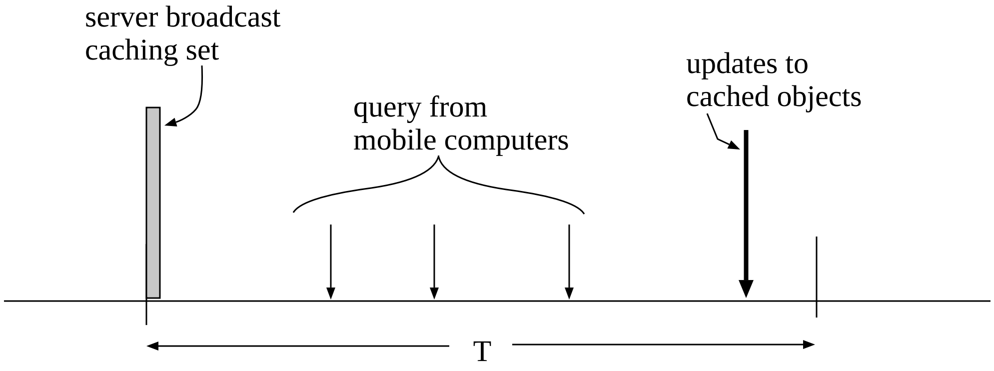



Figure 2: Basic broadcast protocol.

| | | |
|---|---|---|
| $C_w$ | = | write cost for issuing broadcast when a cached object is modified. |
| $BW$ | = | bandwidth within a cell in bits per period of time T. |
| $N$ | = | total number of objects within a cell. |
| $M$ | = | total number of mobile computers within a cell. |

We assume that data is accessed by units of data objects. (These may correspond to disk page units.) We assume that all data objects are of the same size. In the above, $C_r$ is the cost of reading a data object by a MC if the data object is not cached in the MC. This would include the bandwidth requirement for for the data request message to the SC and the message to relay the data from SC to the MC. Note that if data objects are actually of larger variable size, we can divide the large objects into smaller ones, each of our unit size. This justifies our assumption of constant size data objects. $C_w$ is the cost of writing a data object when the data is cached at one or more MCs. It would be the bandwidth requirement for the broadcast of the data value from the SC to the MCs in its cell. Our objective function is:

$$\begin{aligned} \text{M}inimize\ Z &= \sum_i \sum_j \lambda^r_{ij} T C_r (1 - \Gamma_{ij}) + \sum_j (\lambda^w_j T + 1) C_w [1 - \prod_i (1 - \Gamma_{ij})] \\ \text{s.}t\ \sum_j \Gamma_{ij} &\leq B_i \quad \forall i \\ \Gamma_{ij} &= 0\ or\ 1 \quad \forall i, j \end{aligned} \tag{1}$$

In the objective function, we try to minimize the cost of communication by setting the right values of $\Gamma_{ij}$ for each $i$ and $j$. In other words we would like to find out what objects should be cached in each mobile computer in order to minimize the cost. In the function, the term $\prod_i (1 - \Gamma_{ij})$ is 0 if the object $j$ is cached in any MC and equals 1 otherwise. Note that in the term $(\lambda^w_j T + 1)$, the value 1 represents the broadcast which may be necessary at the beginning of the time period if an object is to be cached at some MC which does not contain that object at the moment. For now we assume that the SC simply broadcasts the identities of all the objects that are to be cached.

## 2.1 Complexity of the Problem

In this section, we establish that most likely, there will not be any efficient algorithm for solving the minimization problem as defined by our objective function (1). This means that some kind of heuristic approach will be necessary. Let us call the minimization problem as described in the previous section the *mobile caching* problem

**Theorem** : The mobile caching problem is NP-hard.

**Proof** : First we define a restricted version of the given problem, we shall call it the *restricted mobile caching* problem. The instance and question of this problem are:

- Instance: Given a set of $M$ MC's, each with a buffer size of 1, a set of $N$ data objects, and the values of $\lambda^w$, $\lambda^r$, $C_r$ and $C_w$ as defined for objective function (1). Also given a positive integer $K'$.
- Question: Is there a cache assignment of $\Gamma_{ij}$ as defined for the objective function (1), such that the reduction in the communication cost (as given by equation (1)) as compared to the non-caching case (where all $\Gamma_{ij}$ are zero) is at least $K'$.

We transform the NP-complete problem of *set packing* [8] to the restricted mobile caching problem. Here we give a definition of the set packing problem:

*Definition of Set Packing:*
Instance: Collection $C$ of finite sets, positive integer $K \leq |C|$.
Question: Does $C$ contain at least $K$ mutually disjoint sets?

The transformation is as follows: For a set packing instance $I_1$, let $E$ be the set of all elements $e$ such that $e \in S$ where $S$ is in $C$. From $I_1$ we create a mobile caching instance $I_2$. Let the time period T of $I_2$ be 1. Create one mobile computing unit for each element in $E$. We refer to an MC unit created for an element $e$ as $MC_e$. Let the number of MC units created be $N$. Let the buffer size for each MC unit be 1. Create one data object for each set $S$ in $C$. Let $M$ be the number of objects created. For each set $S$ in $C$, let $S = \{e_1, e_2, ...e_s\}$. Let $D_S$ be the data object created for $S$.

Assign read costs and write costs as follows:
Let $R_{e_iD_S}$ be the value of $\lambda^r_{D_Se_i}C_r$ for object $D_S$ and $MC_{e_i}$.
Let $W_{D_S}$ be the value of $\lambda^w_{D_S}C_w$ for object $D_S$.
Let $C_r$ and $C_w$ be 1 so that $R_{e_iD_S} = \lambda^r_{D_Se_i}$ and $W_{D_S} = \lambda^w_{D_S}$

For each $S$ in $C$, we assign 0 to $R_{e_jD_S}$ for $e_j$ not in $S$. By choosing suitable values of read/write rates and costs, we could ensure that

$$R_{e_1D_S} + R_{e_2D_S} + ...R_{e_sD_S} - W_{D_S} - 1 = c \qquad (2)$$

where $c$ is a positive constant. We would also make sure that $c$ is less than all $R_{e_kK_S}$ for $e_k$ in $S$. For the value of $K$ given in $I_1$, we define $K' = cK$ in $I_2$.

One way to achieve the above is: Let read cost $R_{e_iD_S}$ at each $MC_{e_i}$ in $S$ be 2. Let $c$ = 1, and write cost $W_{D_S}$ be $2s - 2$. Note that $s$ is the number of elements in $S$. Then the above equality (2) holds and $c$ is less than all $R_{e_kK_S}$ for $e_k$ in $S$. Hence $K' = K$. It is easy to verify that $I_2$ is an instance of the restricted mobile cache problem. It is also easy to see that the above transformation has polynomial time complexity.

Note that (2) represents the reduction in communication cost compared to the non-caching case if we cache an object $D_S$ at each $MC_e$ with non-zero $R_{e_jD_S}$. Hence the reduction is $c$ for all objects. With the above set up, if an object is cached then all the MC's with a non-zero read cost for the item must cache the item to achieve minimal overall cost (maximal cost reduction over the non-caching case). This corresponds to choosing a set in $C$ for the mutually disjoint set in the set packing problem. Since buffer size is 1 at each MC, we cannot pick 2 data objects to be cached at an MC such that the read cost for both are non-zero at the MC. This corresponds to not choosing 2 sets that have overlapping elements in the set packing problem $I_1$, which will ensure the property of mutually disjoint sets. Since the reduction in cost is $c$ for all objects in $I_2$, the maximal reduction is obtained by caching the maximal number of data objects in the entire network. This corresponds to choosing the maximal number of sets in $I_1$.

Suppose a set of $\Gamma_{ij}$ values gives maximal communication cost reduction of $G$ over the non-caching case in $I_2$. Suppose $X$ is the set of data objects that are cached in some MC as governed by $\Gamma_{ij}$. If $Y$ is the set of sets in $C$ for which data objects in $X$ are created, then $Y$ corresponds to a maximal set of mutually disjoint sets, and $|Y| = G$. Hence if $|X| \geq K$, then $|Y| \geq K$, or $I_2$ is a YES instance implies $I_1$ is a YES instance. Conversely, if $Y$ is a maximal set of mutually disjoint sets in the set packing problem instance $I_1$, then the set of data objects $X$ created for $Y$ would be a set of cached objects that gives maximal communication cost reduction over the non-cache case. Hence if $I_1$ is a YES instance, then $I_2$ is also a YES instance.

Since the restricted mobile caching problem is a restriction of the mobile caching problem, this completes the proof. ∎

The above shows that the problem is NP-hard even if we restrict the buffer size of each MC to be 1.

# 3 Caching Algorithms

It is assumed in this paper that the local stationary computer (SC) in a cell is the only source of data for mobile computers (MC's). If a data object $j$ is not cached in a mobile computer, a read request will be sent to the local SC for each read operation on object $j$. Suppose a read request contains $b_q$ bytes and the response to the mobile computer contains $b_r$ bytes. Then each read request requires a transmission of $b_r + b_q$ bytes, which is referred to as the cost, $C_r$, of the read operation. Since the rate of read requests generated from mobile computer $i$ on data object $j$ is $\lambda_{ij}^r$, the bandwidth consumed by mobile computer $i$ for object $j$ will be $\lambda_{ij}^r(b_q + b_r)$. Furthermore, if we assume that no data object is cached in any mobile computer in a cell, the total bandwidth required to satisfy all the read requests in a cell will be $\sum_i^M \sum_j^N \lambda_{ij}^r(b_q + b_r)$.

Obviously, the bandwidth required for read requests cannot exceed the bandwidth provided by the wireless network. Therefore, the number of mobile computers that can be supported by a cell, known as the capacity of the cell, is limited. In order to increase the capacity of a cell, the bandwidth required for read requests should be reduced. One of the possible ways is to allow mobile computers to cache some of the frequently used data objects. By using this approach, a read operation on a cached data object can be satisfied immediately without sending a request to the stationary computer. Hence, caching not only saves the bandwidth required for read requests, but also improves the response time for read operations.

However, data objects in stationary computers are continuously being updated. If a data object is cached in some mobile computers in a cell, it is required to broadcast the new value of the data object immediately after each update of the data object. Such a support from the stationary computer is essential to keep the data object cached in mobile computers as up-to-date as possible. It is expected that the structure of a message broadcasted for an update should be similar to that of a response to a read request. Assume that the length of a message for an update is $b_r$, i.e., $C_w = b_r$, and the update rate of data object $j$ is $\lambda_j^w$. Then the bandwidth required to broadcast new values of data object $j$ will be $\lambda_j^w b_r$. Suppose we use a caching scheme which dictates that

$cache(j)$ is the set of mobile computers that will cache data object $j$. The bandwidth consumed by read requests on data object $j$ will be dropped from $\sum_i^M \lambda_{ij}^r(b_q + b_r)$ to $\sum_{i \notin cache(j)} \lambda_{ij}^r(b_q + b_r)$ when caching of data object $j$ is supported; however the bandwidth required for broadcasting new values of the object will be increased from 0 to $\lambda_j^w b_r$. Therefore, the net saving in bandwidth is

$$\sum_{i \in cache(j)} \lambda_{ij}^r(b_q + b_r) - \lambda_j^w b_r.$$

Obviously, if the net saving is negative, then the SC should not support caching of the data object.

The problem now is how to efficiently select data objects to be cached at each mobile computer so as to maximize the saving in bandwidth. However, in Section 2.1, we have proved that this problem is NP-hard. That means it is not quite possible to derive an efficient algorithm to optimize the saving in bandwidth. In order for the system to adapt to the changes of access pattern as quickly as possible, $\mathcal{S}$ must be determined efficiently. In the following sections, two algorithms are proposed. Although the caching decisions obtained from these two algorithms may not be optimal, the algorithms can be run efficiently. Their performance are evaluated in Section 5.

## 3.1 Baseline Algorithm

We assume that given a caching set $\mathcal{S}$, each mobile computer can freely choose a subset of data objects to be cached so as to minimize the number of read requests to its SC. Suppose the maximum number of data objects that can be cached in a mobile computer is $B$. Obviously, the mobile computer will select the top $B$ data objects from $\mathcal{S}$ with respect to the read rates of the data objects. Therefore, the SC can compute the saving in bandwidth for a given $\mathcal{S}$, provided that the read rate and write rate of each data object for each mobile computer is known.

We first propose a baseline algorithm. This algorithm can be used as a reference to compare the performance of other algorithms. The fundamental idea of the baseline algorithm is that the number of data objects in $\mathcal{S}$ is equal to $B$. Therefore, each mobile computer will simply cache every data object in $\mathcal{S}$ in order to minimize the number of read requests to the SC. Based on this property, the saving in bandwidth by adding a data object $j$ in $\mathcal{S}$ can be expressed as:

$$\sum_{i=1}^{M} \lambda_{ij}^r C_r - \lambda_j^w C_w \quad (3)$$

---

FOR $j = 1$ to $N$
    $d[j].gain = \sum_{i=1}^{M} \lambda_{ij}^r (b_q + b_r) - \lambda_j^w b_r$
ENDFOR
$d' \leftarrow Sort(d)$ /* Sort the records with respect to $d[].gain$ */
$\mathcal{S} \leftarrow \{d'[j].id | j \leq B \wedge d'[j].gain > 0\}$

---

Figure 3: Baseline Algorithm

In order to maximize the saving in bandwidth, the baseline algorithm will select the top $B$ data objects (in term of the saving in bandwidth) into $\mathcal{S}$. The baseline algorithm is depicted in Figure 3. Let $d$ be an array of records which contain two fields, namely, gain and id. For each data object $j$, $d[j].id$ is the identity of the data object $j$ and $d[j].gain$ is the saving in bandwidth by including the data object $j$ into $\mathcal{S}$. After $d[j].gain$ is computed for each data object $j$, the array $d$ will be sorted with respect to $d[j].gain$. Then the first $B$ data objects will be selected into $\mathcal{S}$ if their saving in bandwidth are positive. For example, consider the read rate and write rate shown in Tables 1 and 2, where $O_1, O_2, O_3$ and $O_4$ are data objects and $MC_1$ and $MC_2$ are mobile computers. For simplicity, assume that $C_r$ and $C_w$ are 1. Then by Equation 3, the vector $d[].gain$ can be computed easily and is shown in Table 3. Consider the case where $B = 2$, then the baseline algorithm will select the two data objects which will maximize the saving in bandwidth. Therefore, $\mathcal{S} = \{O_2, O_3\}$.

| | $O_1$ | $O_2$ | $O_3$ | $O_4$ |
|---|---|---|---|---|
| $MC_1$ | 4 | 4 | 2 | 1 |
| $MC_1$ | 1 | 4 | 4 | 1 |

Table 1: Read Rate ($\lambda^r$)

| | $O_1$ | $O_2$ | $O_3$ | $O_4$ |
|---|---|---|---|---|
| Write | 1 | 2 | 1 | 1 |

Table 2: Write Rate ($\lambda^w$)

It is not difficult to observe that the complexity of computing the vector $d[].gain$ is $O(MN)$, the complexity of sorting $d$ is $O(NlogN)$ and the complexity of selecting the top $B$ objects is $O(B)$. Since we $B < N$ (otherwise, we can cache the whole database), the complexity is given by $O(N(M + logN))$. In the complexity expression, $N$ can actually

| | $O_1$ | $O_2$ | $O_3$ | $O_4$ |
|---|---|---|---|---|
| Gain | 4 | 6 | 5 | 1 |

Table 3: Saving in Bandwidth ($d[].gain$)

be replaced by $P$, which is the number of data objects that have non-zero values of read rates. This is because we can ignore data objects that are not read by any mobile computers, since definitely they would not be cached. Note that if for all data object $j$, $d[j].gain$ is negative, then it is better not to support caching of any data object. This situation arises when the update rate of the data objects is higher than the aggregate read rate of the data objects.

## 3.2 Greedy algorithm

In the baseline algorithm, the size of $\mathcal{S}$ is limited to $B$, where $B$ is the cache size of each mobile computer. It is expected that the baseline algorithm is suitable for the environment in which the access pattern of each mobile computer is similar, because $\mathcal{S}$ contains the most frequently accessed $B$ objects and these objects are cached in each mobile computer. However, when the access patterns of the mobile computers are different, the set of frequently accessed objects for each mobile computer are different. Therefore, the set of data objects required to be supported by the SC should be larger than $B$. In this case, the performance of the baseline algorithm may not be satisfactory. This observation motivates the design of the greedy algorithm presented in this section. By using the greedy algorithm, the SC supports the caching of more than $B$ data objects and each mobile computer $i$ selects a set $S_i \in \mathcal{S}$ such that $|S_i| = B$ that satisfy its access pattern, i.e., minimize the number of read requests. The greedy algorithm works in this way. First, a set of data objects is obtained by using the baseline algorithm. Then the set is augmented by adding one data object to the set at a time. This process is continued until the additional saving in bandwidth by adding the next data object becomes negative. The algorithm is depicted in Figure 4. The most crucial part of the algorithm is to select in each iteration the data object to be added into the set. When a data object $j$ is added into $\mathcal{S}$, a mobile computer $i$ will decide to drop the data object with the minimum read rate, say $y$ from its cache and then cache $j$, if the read rate of $j$ is larger than the read rate of $y$. In this case $switch[i].flag$ will be set to true and $switch[i].obj$ will be set to $y$. For mobile computer $i$, the net saving in bandwidth for read requests is $\lambda^r_{ij} - \lambda^r_{iy}$. Therefore, the net saving in bandwidth for read requests in the whole cell is

$$(b_q + b_r)\Big(\sum_{\{i|switch[i].flag=TRUE\}} (\lambda^r_{ij} - \lambda^r_{i(switch[i].obj)}\Big).$$

Since, the bandwidth required for broadcasting the updates of $j$ is $\lambda_j^w b_r$, the net saving in bandwidth by including $j$ in $\mathcal{S}$, $d[j].gain$, is

$$(b_q + b_r)\left(\sum_{\{i|switch[i].flag=TRUE\}} (\lambda_{ij}^r - \lambda_{i(switch[i].obj)}^r)\right) - \lambda_j^w b_r.$$

By the essence of greedy algorithms, the data object $a$ with the maximum $d[a].gain$ will be selected to be the next data object included in $\mathcal{S}$. The iteration is repeated until $d[a].gain \leq 0$

---

$\mathcal{S} \leftarrow$ Result from the Baseline Algorithm
REPEAT
  FOR $j = 1$ to $N$
    IF $j \notin \mathcal{S}$
      $d[j].gain \leftarrow (b_q + b_r)(\sum_{\{i|switch[i].flag=TRUE\}}(\lambda_{ij}^r - \lambda_{i(switch[i].obj)}^r)) - \lambda_j^w b_r)$
    ELSE
      $d[j].gain \leftarrow 0$
    ENDFOR
  Let $a$ be $j$ such that $d[j].gain$ is maximum
  IF $d[a].gain > 0$
      $\mathcal{S} \leftarrow \mathcal{S} \cup \{a\}$
  ENDIF
UNTIL $d[a].gain \leq 0$

---

Figure 4: Greedy Algorithm

For example, consider the read rate and write rate shown in Tables 1 and 2. Suppose the current $\mathcal{S}$ is $\{O_2, O_3\}$, hence both $MC_1$ and $MC_2$ will cache $O_2$ and $O_3$. If $O_1$ is added into $\mathcal{S}$, $MC_1$ will cache $O_1$ instead of $O_3$ because $\lambda_{11}^r > \lambda_{13}^r$, but $MC_2$ will not change its caching set because $\lambda_{21}^r < \lambda_{23}^r$. Therefore the saving in bandwidth for read requests is $4 - 2 = 2$. However the bandwidth required for broadcasting the updates of $O_1$ is 1. Hence the net saving in bandwidth by including $O_1$ into $\mathcal{S}$, $d[1].gain = 1$. Similarly, $d[4].gain = -1$, since neither $MC_1$ nor $MC_2$ will cache $O_4$ even if $O_4$ is added into $\mathcal{S}$. Thus, $O_1$ will be included into $\mathcal{S}$ and the algorithm stops in the next iteration.

In the worst case, the greedy algorithm will stop after taking $N - B$ iterations. For each iteration, the complexity of calculating the the vector $d[].gain$ is $O(MN)$. Hence the complexity is given by $O(MN(N - B))$. As for the Baseline Algorithm, in this complexity, $N$ can be replaced by $P$, which is the number of data objects that is read by at least one MC.

# 4 Performance Analysis

In this section, we derive various performance measures and compare the algorithms we have described in the previous section, namely, 1) the non-caching policy (NCP) and, 2) the caching policy (CP). For the class of caching policy, we analyze i) the baseline algorithm (BA) and, ii) the greedy algorithm (GA). Under each class of policy, we are interested in the following performance measures, the expected residual bandwidth ($E[BW_r]$), the expected read operation response time and the system stability condition.

## 4.1 Expected Residual Bandwidth

The expected residual bandwidth ($E[BW_r]$) is an important performance measure because it indicates the available bandwidth to process any requests generated by the read operations of mobile computers which may have just traversed to this cell from neighboring cells. Note that the higher the residual bandwidth, the higher the probability that the system can accept requests from new mobile computer which have just join the cell.

Let $\tilde{b}_q$ and $\tilde{b}_r$ be the non-negative random variables representing the length (in bits) of a query generated by the mobile computers and the reply by the stationary computer within a cell, respectively. Let $f_{b_q}(x)$, $F_{b_q}(x)$, $G^*_{b_q}(s)$ ($f_{b_r}(x)$, $F_{b_r}(x)$, $G^*_{b_r}(s)$) be the density function, probability distribution function and Laplace transform for random variable $\tilde{b}_q$ ($\tilde{b}_r$), respectively. Also, denote by $x$ and $x^{(n)}$ the first and the $n^{th}$ moments for any random variable $\tilde{x}$. For the NCP, the expected residual bandwidth is:

$$E[BW_r^{NCP}] \;=\; BW - \sum_{i=1}^{M}\sum_{j=1}^{N}[\lambda^r_{ij}(b_q + b_r)] \tag{4}$$

where $b_q$ and $b_r$ are the first moment of the length of query and reply.

For caching policies (either the baseline or greedy algorithm), let $\mathcal{S}$ be the set of all cached objects in the system and $\mathcal{S}_i$ be the set of cached object for mobile computer $i$ in a given cell. The expected residual bandwidth is:

$$E[BW_r^c] \;=\; BW - \frac{1}{T}|\mathcal{S}|b_r - \sum_{j\in\mathcal{S}}\lambda^w_j b_r - \sum_{i=1}^{M}\sum_{j\notin\mathcal{S}_i}\lambda^r_{ij}(b_q + b_r) \tag{5}$$

The second term at the right hand side represents the broadcast traffic generated by the cached objects in $\mathcal{S}$ at the beginning of each period, the third term is contributed by all write updates to all cached objects in $\mathcal{S}$. Since these objects in $\mathcal{S}$ are cached by some mobile computers, the system needs to perform invalidation or update to all mobile computers. The third term consists of all the traffic generated by read operations from the mobile computers due to cache missed.

## 4.2 System Response Time and Stability

To derive the read operation response time, we first assume that the processing time at the SC is negligible compare to the delay experience in the communication channel within a cell. This assumption is reasonable because the 1) the processing power at the server is much larger than that of the mobile computers and, 2) the channel bandwidth in the wireless communication platform is quite limited. Notice that any query generated by the mobile computer has to first go through the communication channel, then this query is processed by the stationary computer within the cell. A reply for this query will then be sent by the stationary computer. Again, this reply has to go through the communication channel. Given the dynamics of the query and reply, we model the communication channel within a cell as a $M/G/1$ feedback queue which follows a FCFS scheduling discipline [6]. Notice that although the channel is a fixed rate server with a service rate of $BW$, different classes of jobs (for example, query, reply or broadcast) have different service time distribution because each class has different bit length. In the following, we first derive the Laplace transform of service time of different classes of jobs. These transforms are important because they can be used to obtain various moments of service time in computing the expected response time of different classes of jobs.

Without the loss of generality, let $\tilde{x}$ be a non-negative random variable representing the length, in bits, of a class of job and $\tilde{\tau}$ be the random variable of the service time of this job. We have:

$$\tilde{\tau} = \frac{\tilde{x}}{BW}$$

The following lemma states the Laplace transform of the service time,$B^*_\tau(s)$, of $\tilde{\tau}$:

**Lemma 1** *Let $B^*_\tau(s)$ be the Laplace transform of the service time of a job, it can be expressed as:*

$$B^*_\tau(s) \quad = \quad G^*_x\left(s/BW\right) \tag{6}$$

*where $G^*_x(s)$ is the Laplace transform of random variable $\tilde{x}$.*

**Proof:** please refer to the appendix. ∎

Once we know the Laplace transform of the service time of a job. We can easily compute the expected value (the first moment) as well as the higher moments of the service time by simply evaluation the following:

$$\tau^{(n)} = (-1)^n \left. \frac{d^n B^*_\tau(s)}{ds^n} \right|_{s=0} \qquad n = 1, 2, \ldots \tag{7}$$

Note that in order to derive the expected response time for different policies, we need to compute the first two moments of service times for various class of jobs in the system. We consider the derivation of the NCP and CP separately.

### 4.2.1 Non-caching policy

In this section, we derive the expected response time for a read operation under the non-caching policy. Since the request rates from the mobile computers $\lambda^r_{ij}$ are Poisson. The overall query request arrival rate ($\lambda^q_{nc}$) to the channel is still Poisson, which is expressed as:

$$\lambda^q_{nc} = \sum_{i=1}^{M} \sum_{j=1}^{N} \lambda^r_{ij} \tag{8}$$

The model we consider for the non-caching policy is depicted in Figure 5. The total

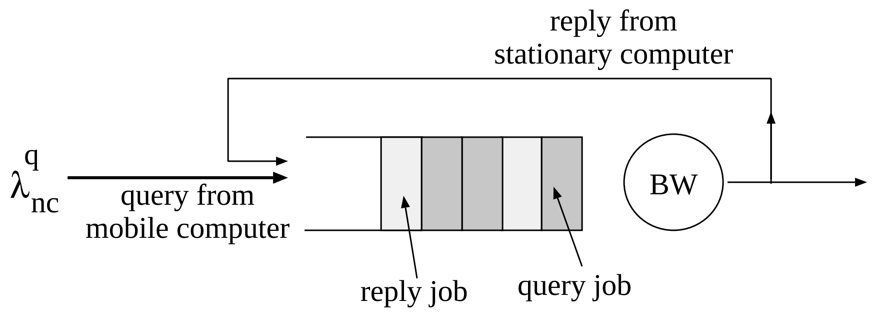



Figure 5: Queuing Model for NCP.

read query from the mobile computers is $\lambda^q_{nc}$. The server, which represents the communication channel, has a fixed service rate of $BW$. The size of each query request is characterized by the density function $f_{b_q}(x)$ and the Laplace transform $G^*_{b_q}(s)$. When a query request finishes its service in the communication channel, the stationary computer within the cell will immediately send out the value of the requested data object into the communication channel. This can be modeled by an immediate feedback after the service of a query request. The size of this feedback job is characterized by the density function $f_{b_r}(x)$ and the Laplace transform $G^*_{b_r}(s)$. Therefore, the response time of a read operation under the non-caching policy, denotes by $T_{nc}$, is the time it takes for the query to go through the communication channel ($T^q_{nc}$), together with the time it takes for the reply to go through the communication channel, $T^r_{nc}$. Mathematically, we have:

$$T_{nc} \;=\; T^q_{nc} + T^r_{nc} \tag{9}$$

Let $t_{b_q}$ and $t^{(2)}_{b_q}$ ($t_{b_r}$ and $t^{(2)}_{b_r}$) be the first and second moments of the service time of a query (reply). Note that given the Laplace transforms of $G^*_{b_q}(x)$ and $G^*_{b_r}(x)$, we can compute the moments of service time based on Lemma 1 and Equation (7).

We start by obtaining the expression of $T^q_{nc}$ by considered a *tagged* query customer. Let $N_q$ and $N_r$ represent the expected number of query and reply jobs in the system (both in the waiting queue and in the server) at arbitrary epoch in time. Note that the aggregated traffic from the mobile computers within the cell is a Poisson process and hence PASTA property [17] applies. Now, consider the expected amount of work that needs to be finished by the server before this *tagged* query customer receives its service. The expected amount of work consists of two parts:

- the expected amount of work waiting in the queue only found by the query job upon arrival. Based on the FCFS discipline, this waiting work has to be handled before the service of the tagged query job. The work is equal to the expected number of query jobs waiting in the queue multiply by the query expected service time and the expected number of reply jobs waiting in the queue multiply by the reply expected service time. Therefore, the expression is :

  $$(N_q - \lambda^q_{nc} t_{b_q}) t_{b_q} + (N_r - \lambda^q_{nc} t_{b_r}) t_{b_r}$$

  The expression in the parenthesis represents the number of jobs (either query or reply) waiting in the communication queue only.

- the expected amount of work currently in service. This is equal to the expected remaining service time for either query or reply job, weighted by their system utilization factors. The expression is:

  $$\rho^q_{nc} \frac{t^{(2)}_{b_q}}{2t_{b_q}} + \rho^r_{nc} \frac{t^{(2)}_{b_r}}{2t_{b_r}}$$

  where $\rho^q_{nc} = \lambda^q_{nc} t_{b_q}$ and $\rho^r_{nc} = \lambda^q_{nc} t_{b_r}$.

Therefore, $T^q_{nc}$ is equal to the expected amount of work in front of the tagged arriving query customer, together with its average service time:

$$T^q_{nc} = (N_q - \lambda^q_{nc} t_{b_q}) t_{b_q} + (N_r - \lambda^q_{nc} t_{b_r}) t_{b_r} + \rho^q_{nc} \frac{t^{(2)}_{b_q}}{2t_{b_q}} + \rho^r_{nc} \frac{t^{(2)}_{b_r}}{2t_{b_r}} + t_{b_q} \tag{10}$$

$T^r_{nc}$, which is the expected response time of the reply job, consists of: 1) the expected amount of workload for the *old* jobs (these are jobs which were of query-typed during the tagged query job arrival and are now reply-typed job), 2) the expected amount of workload for all new query jobs which arrived during the period $T^q_{nc}$ and, 3) the expected service time of the reply job. Therefore, we have:

$$T^r_{nc} = N_q t_{b_r} + \lambda^q_{nc} T^q_{nc} t_{b_q} + t_{b_r} \tag{11}$$

Based on the Little's result[14], we have:

$$N_q = \lambda^q_{nc} T^q_{nc} \quad ; \quad N_r = \lambda^q_{nc} T^r_{nc}$$

Apply these two expressions to Equations (10) and (11). We have:

$$\begin{aligned} T^q_{nc} &= (\lambda^q_{nc} T^q_{nc} - \lambda^q_{nc} t_{b_q}) t_{b_q} + (\lambda^q_{nc} T^r_{nc} - \lambda^q_{nc} t_{b_r}) t_{b_r} + \rho^q_{nc} \frac{t^{(2)}_{b_q}}{2t_{b_q}} + \rho^r_{nc} \frac{t^{(2)}_{b_r}}{2t_{b_r}} + t_{b_q} \\ T^r_{nc} &= \lambda^q_{nc} T^q_{nc} t_{b_r} + \lambda^q_{nc} T^q_{nc} t_{b_q} + t_{b_r} \end{aligned}$$

Now we have two linear equations with two unknowns. Simplifying, we have:

$$T^q_{nc} = \frac{\rho^q_{nc} \frac{t^{(2)}_{b_q}}{2t_{b_q}} + \rho^r_{nc} \frac{t^{(2)}_{b_r}}{2t_{b_r}} + t_{b_q}(1 - \lambda^q_{nc} t_{b_q})}{1 - \lambda^q_{nc} t_{b_q} - (\lambda^q_{nc})^2 t_{b_r} (t_{b_r} + t_{b_q})} \tag{12}$$

$$T^r_{nc} = \lambda^q_{nc} (t_{b_r} + t_{b_q}) T^q_{nc} + t_{b_r} \tag{13}$$

Based on classic queuing theory [13], the stability condition for the NCP is:

$$\rho^q_{nc} + \rho^r_{nc} < 1 \tag{14}$$

### 4.2.2 Caching policies

For caching policies[1], note that some of the read operations will be satisfied within the cache of the MC. In this case, we assume that the response time of this read operation is zero (that is, immediate access to the data). If the data item requested is not available in

[1]Note that the derivation for the baseline and greedy algorithm is the same. The only difference is the elements in $\mathcal{S}$ and $\mathcal{S}_i$ for each mobile computer $i$ within a given cell.

the local cache, these read operations have to go through the communication channel to access the data from the stationary computer within the cell. In this case, the response time of this read operation is greater than zero. Another important thing to note is that, for the caching policy, there are three types of traffic going through the communication channel. They are

- any read operations generated by the mobile computer $i$ such that the data item is not in $\mathcal{S}_i$.
- if there is any new update to cached data in any mobile computer, the new data value has to be broadcast to all mobile computing units.
- periodic broadcast from the stationary computer for all data objects in the caching set $\mathcal{S}$. We model the time period as an exponential random variable with mean $T$. Since this broadcast is performed on an average every $T$ time units. Therefore, the average rate of broadcast is $\lambda_b = 1/T$.

Let $h$ be the overall probability of cache hit of all mobile computers within a given cell. We have:

$$h = \frac{\sum_{i=1}^{M} \sum_{j \in \mathcal{S}_i} \lambda_{ij}^r}{\sum_{i=1}^{M} \sum_{j=1}^{N} \lambda_{ij}^r} \quad (15)$$

Let $T_c$ be the expected response time for a read operation under the caching policy. It can be expressed as:

$$T_c = (1-h)\left[T_c^q + T_c^r\right] \quad (16)$$

where $T_c^q$ is the expected response time of query request going through the communication channel and $T_c^r$ is the expected response time of reply from the stationary computer going through the communication channel.

The model we consider for the caching policy is depicted in Figure 6. The traffic generated by queries from all the MC's within a cell is a Poisson process with rate $\lambda_c^q$ and it is equal to:

$$\lambda_c^q = \sum_{i=1}^{M} \sum_{j \notin \mathcal{S}_i} \lambda_{ij}^r \quad (17)$$

The size of each query is characterized by the density function $f_{b_q}(x)$ and the Laplace transform $G_{b_q}^*(s)$. The rate of updates to each cached object is $\lambda_u$ and it is equal to:

$$\lambda_u = \sum_{j \in \mathcal{S}} \lambda_j^w \quad (18)$$

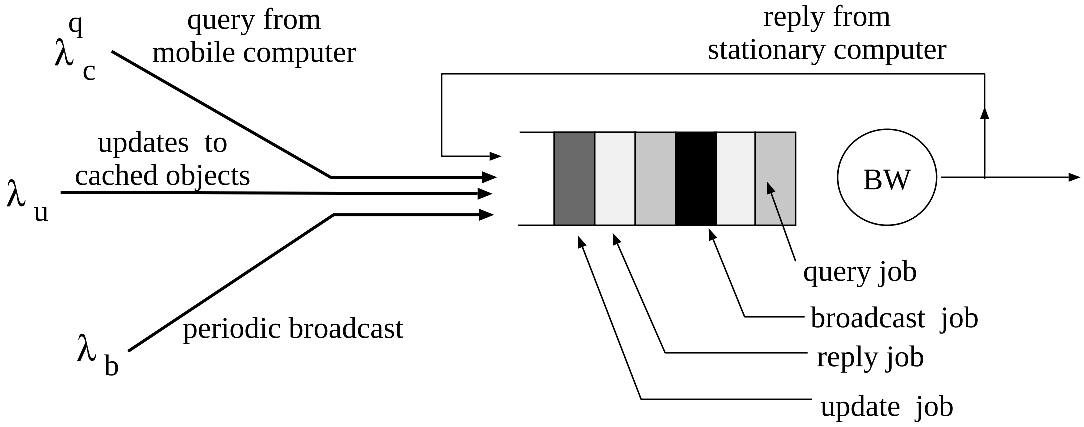



Figure 6: Queuing Model for CP.

and the size of each updates is characterized by the density function $f_{b_r}(x)$ and the Laplace transform $G^*_{b_r}(s)$. Note that this is the same distribution for the size of reply job from the stationary computer for each query request. The periodic broadcast has a mean rate of $1/T$ and the size of each periodic broadcast is a function of the size of the caching set $\mathcal{S}$, denotes by $|\mathcal{S}|$, and it has the Laplace transform of :

$$G^*_{\mathcal{S}}(s) \quad = \quad \prod_{i=1}^{|\mathcal{S}|} G^*_{b_r}(s) \quad = \quad [G^*_{b_r}(s)]^{|\mathcal{S}|} \tag{19}$$

Given this transform, we can obtain the first two moments of the broadcast service time, $t_{b_b}$ and $t^{(2)}_{b_b}$, by applying Lemma 1 and Equation (7).

Using a similar approach as we did in deriving the expected response time for the non-caching policy, we can derive the expected response for each class of jobs going through the communication channel. Let $N_q$, $N_r$, $N_b$ and $N_u$ representing the expected number of query, reply, broadcast and updates in the server at any arbitrary epoch in time, respectively. Now consider a *tagged* customer that is either of query, broadcast or update type. Since all these three classes of arrival are Poisson processes and therefore, the PASTA property [17] applies. The expected unfinished work found by the tagged customer upon its arrival are 1) the expected amount of workload of all four types of jobs in the waiting queue and, 2) the expected remaining service time for all four types of jobs in the server. Together with the expected service time of the tagged job, we can write down the expected response time for these three classes of customers as:

$$\begin{aligned} T^q_c \quad = \quad & (N_q - \lambda^q_c t_{b_q})t_{b_q} + (N_r - \lambda^q_c t_{b_r})t_{b_r} + (N_u - \lambda_u t_{b_r})t_{b_r} + (N_b - \frac{1}{T}t_{b_b})t_{b_b} + \\ & \rho^q_c \frac{t^{(2)}_{b_q}}{2t_{b_q}} + \rho^r_c \frac{t^{(2)}_{b_r}}{2t_{b_r}} + \rho^u_c \frac{t^{(2)}_{b_r}}{2t_{b_r}} + \rho^b_c \frac{t^{(2)}_{b_b}}{2t_{b_b}} + t_{b_q} \end{aligned} \tag{20}$$

$$
\begin{aligned}
T_c^u &= (N_q - \lambda_c^q t_{b_q})t_{b_q} + (N_r - \lambda_c^q t_{b_r})t_{b_r} + (N_u - \lambda_u t_{b_r})t_{b_r} + (N_b - \frac{1}{T}t_{b_b})t_{b_b} + \\
&\qquad \rho_c^q \frac{t_{b_q}^{(2)}}{2t_{b_q}} + \rho_c^r \frac{t_{b_r}^{(2)}}{2t_{b_r}} + \rho_c^u \frac{t_{b_r}^{(2)}}{2t_{b_r}} + \rho_c^b \frac{t_{b_b}^{(2)}}{2t_{b_b}} + t_{b_r} \qquad (21) \\
T_c^b &= (N_q - \lambda_c^q t_{b_q})t_{b_q} + (N_r - \lambda_c^q t_{b_r})t_{b_r} + (N_u - \lambda_u t_{b_r})t_{b_r} + (N_b - \frac{1}{T}t_{b_b})t_{b_b} + \\
&\qquad \rho_c^q \frac{t_{b_q}^{(2)}}{2t_{b_q}} + \rho_c^r \frac{t_{b_r}^{(2)}}{2t_{b_r}} + \rho_c^u \frac{t_{b_r}^{(2)}}{2t_{b_r}} + \rho_c^b \frac{t_{b_b}^{(2)}}{2t_{b_b}} + t_{b_b} \qquad (22)
\end{aligned}
$$

where $\rho_c^q = \lambda_c^q t_{b_q}$, $\rho_c^r = \lambda_c^q t_{b_r}$, $\rho_c^u = \lambda_u t_{b_r}$ and $\rho_c^b = \frac{1}{T} t_{b_b}$.

To compute $T_c^r$, which is the expected expected response time of a reply job, it consists of the expected amount of workload for the *old* jobs (these are jobs which were of query-typed during the tagged query customer arrival and are now reply-typed) as well as new jobs (query, broadcast or updates) which arrived during the period $T_c^q$. Therefore, we have:

$$
T_c^r = N_q t_{b_r} + \lambda_c^q T_c^q t_{b_q} + \lambda_u T_c^q t_{b_r} + \frac{1}{T} T_c^q t_{b_b} + t_{b_r} \qquad (23)
$$

Based on the Little's result[14], we have the following relationships:

$$
N_q = \lambda_c^q T_c^q \;\; ; \;\; N_r = \lambda_c^q T_c^c \;\; ; \;\; N_u = \lambda^u T_c^r \;\; ; \;\; N_b = \frac{1}{T} T_c^b \;\; ;
$$

Apply these four expressions to the response time expressions, we have:

$$
\begin{aligned}
T_c^q &= (\lambda_c^q T_c^q - \lambda_c^q t_{b_q})t_{b_q} + (\lambda_c^q T_c^r - \lambda_c^q t_{b_r})t_{b_r} + (\lambda_u T_c^u - \lambda_u t_{b_r})t_{b_r} + \\
&\qquad (\frac{1}{T}T_c^b - \frac{1}{T}t_{b_b})t_{b_b} + \rho_c^q \frac{t_{b_q}^{(2)}}{2t_{b_q}} + \rho_c^r \frac{t_{b_r}^{(2)}}{2t_{b_r}} + \rho_c^u \frac{t_{b_r}^{(2)}}{2t_{b_r}} + \rho_c^b \frac{t_{b_b}^{(2)}}{2t_{b_b}} + t_{b_q} \qquad (24) \\
T_c^u &= (\lambda_c^q T_c^q - \lambda_c^q t_{b_q})t_{b_q} + (\lambda_c^q T_c^r - \lambda_c^q t_{b_r})t_{b_r} + (\lambda_u T_c^u - \lambda_u t_{b_r})t_{b_r} + \\
&\qquad (\frac{1}{T}T_c^b - \frac{1}{T}t_{b_b})t_{b_b} + \rho_c^q \frac{t_{b_q}^{(2)}}{2t_{b_q}} + \rho_c^r \frac{t_{b_r}^{(2)}}{2t_{b_r}} + \rho_c^u \frac{t_{b_r}^{(2)}}{2t_{b_r}} + \rho_c^b \frac{t_{b_b}^{(2)}}{2t_{b_b}} + t_{b_r} \qquad (25) \\
T_c^b &= (\lambda_c^q T_c^q - \lambda_c^q t_{b_q})t_{b_q} + (\lambda_c^q T_c^r - \lambda_c^q t_{b_r})t_{b_r} + (\lambda_u T_c^u - \lambda_u t_{b_r})t_{b_r} + \\
&\qquad (\frac{1}{T}T_c^b - \frac{1}{T}t_{b_b})t_{b_b} + \rho_c^q \frac{t_{b_q}^{(2)}}{2t_{b_q}} + \rho_c^r \frac{t_{b_r}^{(2)}}{2t_{b_r}} + \rho_c^u \frac{t_{b_r}^{(2)}}{2t_{b_r}} + \rho_c^b \frac{t_{b_b}^{(2)}}{2t_{b_b}} + t_{b_b} \qquad (26) \\
T_c^r &= \lambda_c^q T_c^q t_{b_r} + \lambda_c^q T_c^q t_{b_q} + \lambda_u T_c^q t_{b_r} + \frac{1}{T} T_c^q t_{b_b} + t_{b_r} \qquad (27)
\end{aligned}
$$

Now we have four linear equations with four unknowns which can easily be solved to obtain the expected response time of a read operation under the caching policy, which is expressed in Equation (16). Based on classic queuing theory [13], the stability condition for the caching policy is:

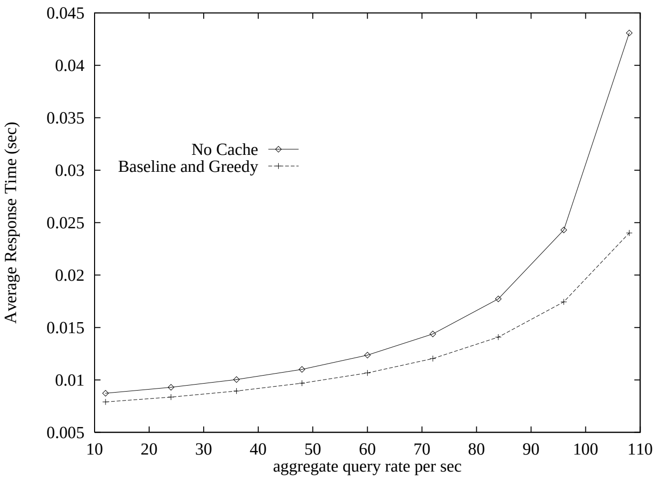
$$
\rho_c^q + \rho_c^r + \rho_c^u + \rho_c^b < 1 \qquad (28)
$$

# 5 Result

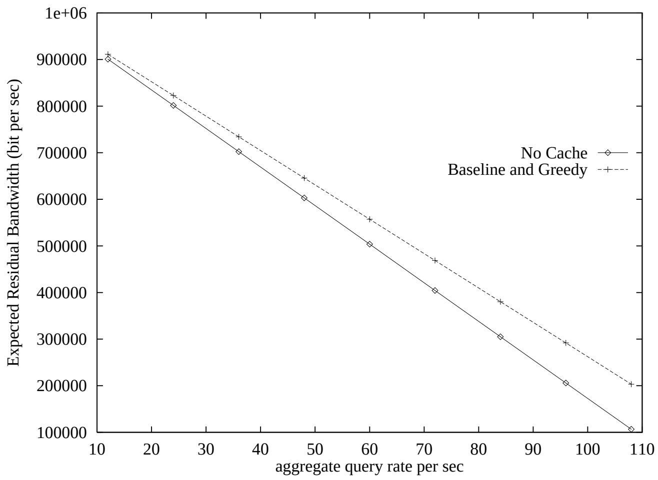
In this section, we carry out different experiments to evaluate the performance gain of our algorithms. We assume that each stationary computer manages 300 data objects, each data object occupies 1 page (which contains 1K bytes). Each mobile computer can cache up to 30 data objects. Each query request generated by a mobile computer has a size of 10 bytes. We also assume that there is constant write traffic to each of the data objects, the aggregate write traffic is 10 requests per second and the targets of this write traffic is evenly distributed at all data objects. The communication bandwidth, in all experiments, is fixed at $1 \times 10^6$ bps. In the first four sets of experiments, the number of mobile computers within a cell is fixed at 50 and the system load is gradually increased by increasing the aggregate read rate within a cell. The objective of these experiments is to study the performance of the proposed caching algorithms under different access patterns. In the last set of experiments, the aggregate read rate for each mobile computer is fixed. The system load is increased by increasing the number of mobile computers within a cell. The objective of this set of experiments is to study how much the capacity of a cell is increased by adopting the proposed caching algorithms.

For the first set of experiments, we assume all mobile computers are running the same application and so the query rates to all data objects are the same. That is, each $\lambda^r_{ij}$ is the same. In Figure 7, we show the expected query response time and the expected residual bandwidth. Since there is no variation in the access rates of different data objects, the caching set $\mathcal{S}$ (the set of objects cached by one or more MC) is the same for the baseline and the greedy algorithm. In this experiment, we see that the improvement of the expected query response time over the non-caching policy ranges from around 10% to 28%; while the expected residual bandwidth has an improvement from 1.2 % to 47 %. Notice that as the system utilization increases (or higher query traffic), caching policy has a higher performance gain.

In the second set of experiments, we still assume that all mobile computers are working on the same application but the data objects are accessed based on a 80/20 access model. That is, 80% of the aggregate query traffic is directed to 20% of the data objects. Mathematically, we have $\lambda^r_{ij} = \lambda^r_{k,j}$ for $1 \leq i \leq k \leq N_m$ and without the loss of generality, we assume the frequently accessed data (*popular objects*) are the first 60 data objects. Note that since all mobile computers are working on the same application, therefore, the caching set $\mathcal{S}$ of the baseline and the greedy algorithm is the same. Figure 8 illustrates the performance gain in both the expected query response time and the expected residual bandwidth. For the expected query response time, we observe a performance improvement from 40% to 80%. For the expected residual bandwidth, we observe a performance improvement ranging from 4% to 340%. Again, as the system utilization increases, caching policy has a higher performance gain.

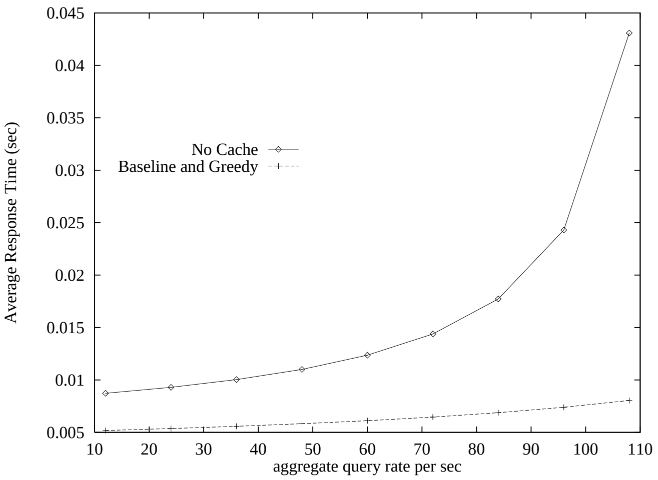



(a) Expected Response Time

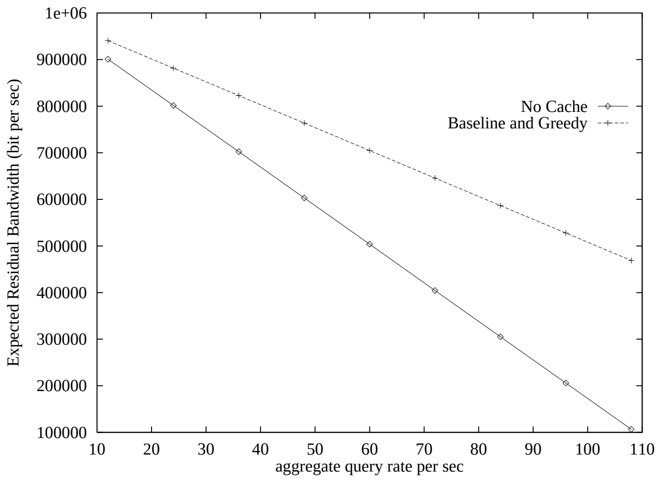



(b) Expected Residual Bandwidth

Figure 7: Case 1: Uniform Traffic



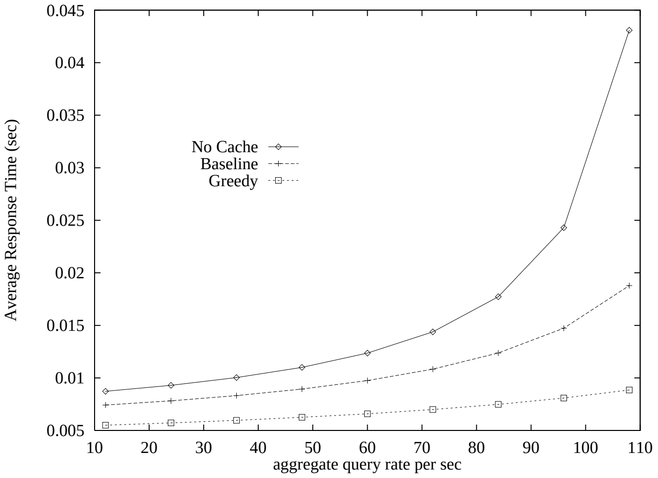

(a) Expected Response Time



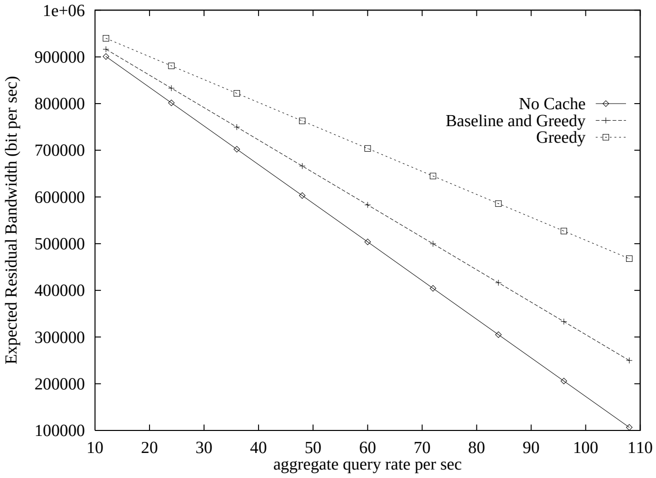

(b) Expected Residual Bandwidth

Figure 8: Case 2: 80/20 Access Model

In the third set of experiments, we partition mobile computers into three classes and each class is working on a different application. For each application, we assume that the data objects are accessed based on a 80/20 access model. Without the loss of generality, we assume that data object 1 to data object 60 are the *popular* objects in first application. Data object 101 to data object 160 are the *popular* data objects in the second application and data object 201 to data object 260 are the *popular* objects for the third application. This implies that there is no overlapping among the sets of popular objects for the three different applications. For each application, the query rate pattern is the same for all mobile computers. Notice that in this situation, the baseline algorithm and the greedy algorithm will have different caching sets $\mathcal{S}$. Figure 9 illustrates the performance gain in both the expected query response time and the expected residual bandwidth. As we can observe, the greedy algorithm has the best performance gain. For the expected query response time, the greedy algorithm has a performance improvement ranging from 36% to 79% (25% to 53%) compared to the non-caching algorithm (the baseline algorithm). For the expected residual bandwidth, the greedy algorithm has a performance gain ranging from 5% to 339% (2.5% to 87%) compared to the non-caching algorithm (the baseline algorithm). Again, as the system utilization increases, the caching policy has a higher performance gain.



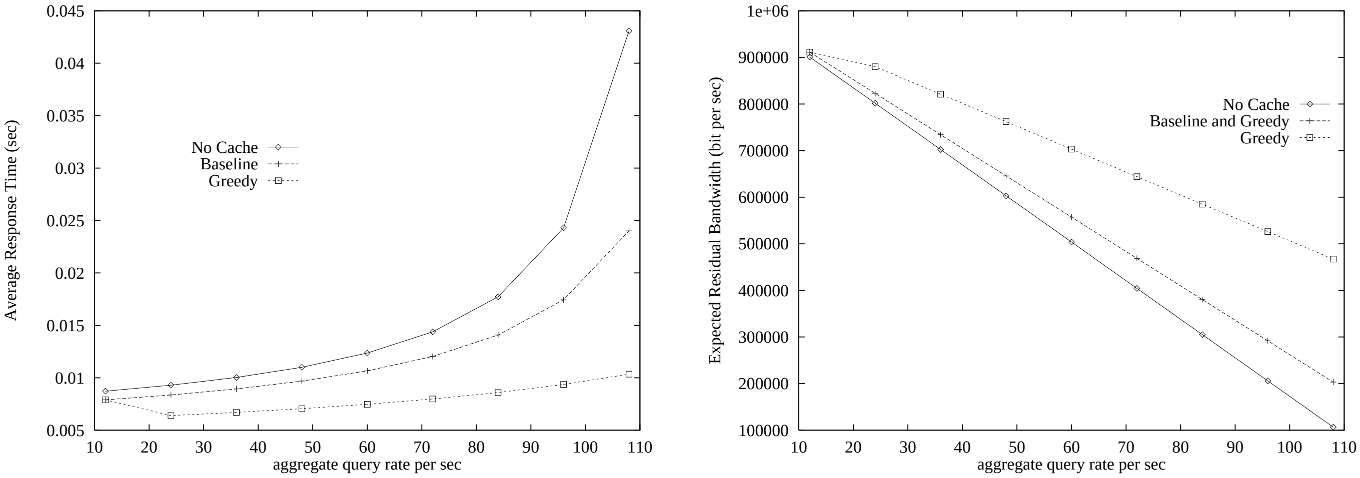

(a) Expected Response Time




(b) Expected Residual Bandwidth

Figure 9: Case 3: 80/20 Access Model; Independent Applications

In the fourth set of experiments, each mobile computer will access data objects based on the 80/20 access model. Unlike the previous experiments, the *popular* objects of different MC's may overlap. The popular data objects for mobile computer $i$ is from $(i*6) MOD\, 300)$ to $((i*6+60) MOD\, 300)$. Notice that in this situation, the greedy algorithm might have the same caching set $\mathcal{S}$ as the baseline algorithm. This is especially true when the system utilization is low. But as we increase the aggregate read rate, the caching set $\mathcal{S}$ of these two algorithms will become different. Figure 10 illustrates the performance gain in both the expected query response time and the expected residual bandwidth. The greedy algorithm, in this case, has the best performance gain. For the expected query response time, the greedy algorithm has a performance improvement up to 80% (57%) comparing to the non-caching algorithm (the baseline algorithm). For the expected residual bandwidth, the greedy algorithm has a performance gain up to 338% (130%) comparing to the non-caching algorithm (the baseline algorithm).




(a) Expected Response Time (b) Expected Residual Bandwidth

Figure 10: Case 4: 80/20 Access Model; Overlapped Applications

In the last set of experiments, we assume each mobile computer uses the 80/20 access model to access data objects. For mobile computer $i$, its popular objects are ranging from $(i*6) MOD\, 300)$ to $((i*6+60) MOD\, 300)$. Each mobile computer has an aggregate read rate of 2.4 per sec. That is for each mobile computer $i$, $\sum_{j=1}^{300} \lambda_{ij}^{r} = 2.4$. The objective of this experiment is to slowly increase the number of mobile computers within a cell and see how many mobile computers each algorithm can support before the system saturates. Figure 11 illustrates the expected response time as we increase the number of mobile computers. It shows that if we want all queries from the mobile computers to have an expected response time of 0.011 secs, the greedy algorithms can support up to 50 mobile computers; while the non-caching policy can only support 20 mobile computers within a given cell. That means the capacity of the cell is increased by 2.5 times. If the expected response time of queries from all mobile computers is 0.042 secs, the greedy algorithm can support up to 63 mobile computers; while the non-caching policy can only support 45 mobile computers. This shows that the caching policy, especially the greedy algorithm, can have a much better performance than the non-caching algorithm.

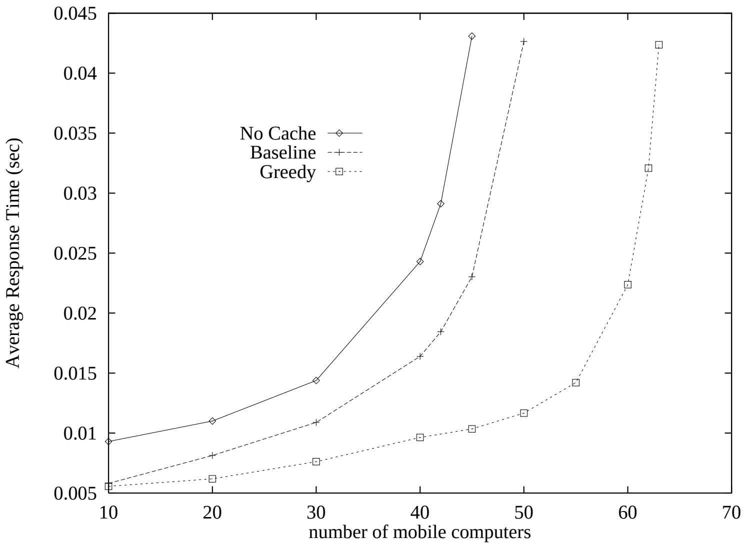



Figure 11: Response time vs. number of mobile computers

# 6 Conclusion

Caching is a significant issue in a mobile computing environment, since the bandwidth of a wireless network is usually very limited as compared to that of a static network. Caching not only reduces the contention on the narrow bandwidth of a wireless channel, but also improves the response time for read operations. When a data object is cached, a read operation on the data object can be satisfied immediately without sending a request to the stationary computer. However, the stationary computer is required to broadcast the new value of the data object when it is updated. In other words, the saving in bandwidth for read operations is at the expense of increasing the bandwidth required for data broadcast. Therefore, the set of data objects, $\mathcal{S}$, that is allowed to be cached in mobile computers must be carefully selected so as to maximize the saving in bandwidth.

However, it has been shown in Section 2.1 that this problem is NP-hard. In order for the system to adapt the changes of access pattern in real time, some heuristic algorithms must be used. In this paper, two novel caching algorithms, baseline algorithm and greedy algorithm, are proposed to select $\mathcal{S}$. Although the $\mathcal{S}$ selected from these two algorithms may not be optimal, the algorithms can be run efficiently. It has been shown in Section 5 that the proposed algorithms outperform the non-caching policy in terms of operation response time and saving in bandwidth under various access patterns. In addition, the capacity of the system can be significantly improved when the caching algorithms are used.

It is observed from the experiments that when the aggregate read rate for each data object is less than its corresponding update rate, it is better not to cache any data objects. In this case, the proposed caching algorithms will set $\mathcal{S} = \emptyset$. In general, a caching policy perform well under skew access pattern, i.e., a large portion of accesses are directed to a small portion of data objects. The performance of the baseline algorithm is the best when the access pattern of each mobile computer is similar, i.e, each mobile computer will decide to cache the same set of data objects. Under this situation, the greedy algorithm and the baseline algorithm will produce the same result. The greedy algorithm gradually outperforms the baseline algorithm when the diversity of the access pattern increases. Another contribution of this paper is that a general performance analysis model is derived in Section 4. This model is general enough to evaluate the performance of other caching policies.

# References


[1] Rafael Alonso, Henry F. Korth. Database System Issues in Nomadic Computing. In *Proceedings of the 1993 SIGMOD Conference*, Washington, D.C., May 1993.

[2] Daniel Barbara, Hector Garcia-Molina. Replicated Data Management in Mobile Environments: Anything New Under the Sun? In *Proceedings of the IFIP Conference on Applications in Parallel and Distributed Computing*, April 1994.

[3] D. Barbara, T. Imielinski. *Sleepers and Workaholics: Caching Strategies in Mobile Environments.* SIGMOD, 1994.

[4] B. R. Badrinath, Arup Acharya, Tomasz Imielinski. Structuring Distributed Algorithm for Mobile Hosts. In *Proceedings of the 14th Intl. Conf. on Distributed Computing System*, June 1994.

[5] B. R. Badrinath, Tomasz Imielinski. Wireless Computing: Challenges in Data Management. *Communications of the ACM*, 1994.

[6] J.L. van den Berg, O.J. Boxma, W.P. Groenendijk. *Sojourn Times in the M/G/1 Queue with Deterministic Feeback*, Stochastic Models,5(1), pp. 115-129, 1989.

[7] George H. Forman, John Zahorjan. *The Challenges of Mobile Computing. IEEE Computer*, 27(6), April 1994.

[8] M. Garey, D. Johnson. *Computers and Intractability. A Guide to the Theory of NP-Completeness W.H. Freeman and Co.* 1979

[9] Y. Huang, P. Sistla, O. Wolfson. *Data Replication for Mobile Computers.* SIGMOD, 1994.

[10] Tomasz Imielinski, B. R. Badrinath. Mobile Wireless Computing: Solutions and Challenges in Data Management. Technical Report DCS-TR-296 and WINLAB-TR-49, Rutgers, Dept. of Comp. Science, 1993.

[11] Tomasz Imielinksi, B. R. Badrinath. Wireless Mobile Computing: Challenges in Data Management. Technical Report DCS-TR-296 and WINLAB TR-49, Rutgers, Dept. of Comp. Sciences, 1993.

[12] T. Imielinski, S. Viswanathan, B.R. Badrinath. *Energy Efficient Indexing on Air.* SIGMOD, 1994.

[13] L. Kleinrock. *Queueing Systems. Volume I: Theory* Wilely Internation, 1975.

[14] J.D.C. Little. *A Proof of the Queueing Formula* $L = \lambda W$. Operations Research, 9, page 383-387. 1961.

[15] Evaggelia Pitoura, Bharat Bhargava. Dealing with Mobility: Issues and Research Challenges. Technical Report TR-93-070, Purdue University, Dept. of Comp. Sciences, 1993.

[16] M. Satyanarayanan, James J. Kistler, Lily B. Mummert, Maria R. Ebling, Puneet Kumar, Qi Lu. Experience with Disconnected Operation in a Mobile Computing Environment. In *Proceedings of the 1993 USENIX Symposium on Mobile and Location-Independent Computing*, Cambridge, MA, August 1993.

[17] R.W. Wolff. *Poisson arrivals see time averages (PASTA).* Journal of Operation Research, 30, page 223-231. 1982.

# Appendix

**Lemma 1.1** *Let* $B_\tau^*(s)$ *be the Laplace transform of the service time of a job with* $\tilde{x}$ *representing its length in bit.* $B_\tau^*(s)$ *can be expressed as:*

$$B_\tau^*(s) = G_x^*\left(s/BW\right)$$

*where* $G_x^*(s)$ *is the Laplace transform for* $\tilde{x}$*.*

**Proof:** The probability distribution function of $\tilde{\tau}$ is:

$$\begin{aligned} F_\tau(y) &= \text{Prob}[\tilde{\tau} \leq y] \;=\; \text{Prob}[\tilde{x} \leq y \cdot BW] \\ &= F_x(y \cdot BW) \end{aligned}$$

The probability density function of $\tilde{\tau}$ is:

$$\begin{aligned} f_\tau(y) &= \frac{d\,F_\tau(y)}{dy} \;=\; \frac{d\,F_x(y \cdot BW)}{d(y \cdot BW)} \frac{d(y \cdot BW)}{dy} \\ &= BW \cdot f_x(y \cdot BW) \end{aligned}$$

The Laplace transform of random variable $\tilde{\tau}$:

$$B_\tau^*(s) = \int_0^\infty BW \cdot f_x(y \cdot BW) e^{-sy} dy$$

We perform variable substitution by letting $z = y \cdot BW$ and we have:

$$\begin{aligned} B_\tau^*(s) &= \int_0^\infty BW f_x(z) e^{-\frac{s}{BW} z} \frac{dz}{BW} \\ &= \int_0^\infty f_x(z) e^{-\frac{s}{BW} z} dz \;=\; G_x^*\left(s/BW\right) \end{aligned}$$

∎