Citation:
Citro, S, McGovern, J and Ryan, C 2005, 'An efficient consistency management algorithm for real-time mobile collaboration', in K.-Y. Cai et al. (ed.) Proceedings of the Fifth International Conference on Quality Software (QSIC2005), Melbourne, 19-20 September 2005.

An Efficient Consistency Management Algorithm for Real-Time Mobile Collaboration

Sandy Citro, Jim McGovern, Caspar Ryan
School of Computer Science and Information Technology
RMIT University
Melbourne, Australia
scitro@cs.rmit.edu.au, jim.mcgovern@rmit.edu.au, caspar@cs.rmit.edu.au

Abstract

Real time mobile collaboration involves two or more co-workers operating concurrently on a shared document using independent mobile devices. The replicated architecture is attractive for such applications since it does not rely on a central server and a user can continue to work on his or her own local document replica even during disconnection period. Several consistency management algorithms have been proposed, however the resource usage of such algorithms, which is critical in a mobile environment, has not been formally studied.

Mobile devices are constrained in terms of memory and processing power, and operate in networking environments with limited bandwidth and transient connectivity. Therefore, algorithms that use resources more effectively will improve the quality of the user experience in a mobile environment. ISO 9126-1 [3] considers software to be efficient if it provides a balance between performance, and resource utilisation while performing its function.

Therefore, this paper evaluates the efficiency of existing techniques, and proposes a more efficient consistency management algorithm. The new algorithm leverages existing techniques which are shown to be efficient and incorporates a novel history management strategy called partial history copy. Different combinations of these techniques are tested and compared to determine which one is most efficient and thus suitable for mobile usage.

1. Introduction

Real time collaboration involves two or more users, at different sites, concurrently working on a shared workspace such as a UML diagram, a multimedia artefact or a text based document. In a collaboration session, users work on a local device which communicates with the individual devices of other users via message passing. Each user interacts with the shared document (as it appears on his or her device) with changes propagated to other users as soon as possible so as to reduce the possibility of update conflicts.

The architecture of a real-time collaboration system can be either centralised or replicated.

In a centralised architecture, only a single copy of the document exists on a central server, with participants updating it directly in a synchronous manner. This approach does however have a number of drawbacks. Firstly, a central server must be present and running at all times, thus introducing a single point of failure whereby the entire collaboration session ceases when the server is down. Secondly, if an individual device is unable to connect to the server, whether due to total network failure or low bandwidth or sporadic disconnection, that user cannot participate in the session. Finally, depending on the implementation, the network usage may be high since all operations must be directed through a central server.

In a replicated architecture, a user interacts with the local document replica by applying local operations to the document, for example, an insertion operation is generated when a character is typed in a text based document. This operation is then broadcast directly to the other participants as a *remote operation*, which is processed at the remote sites before being applied to their document replicas to ensure that operations are executed in such a way so as to preserve the original intention of the user who generated the operation. In this architecture, there is no central server and each device has a replica of the document. Therefore, there is no single point of failure and if a site is disconnected, that user can continue working on his or her own local replica while other sites collaborate with all of the sites to which they have connectivity. Upon reconnection, the previously disconnected site re-synchronise

with other sites to bring its document up to date. This ability to work without a central server, and to operate during disconnection, makes the replicated architecture attractive for mobile groupware. However, implementing replicated consistency management on resource constrained mobile devices is challenging, since existing algorithms [1, 4, 9–11, 13, 15, 16] do not explicitly consider resource utilisation in terms of processing power, memory usage and the resultant effect on power consumption.

ISO 9126-1 [3] considers efficiency as a quality attribute comprising the capability of software to provide appropriate response and processing times (*performance* sub-characteristics) and the capability of software to use appropriate amounts and types of resource during its execution (*resource utilisation* sub-characteristics). Therefore, a consistency management algorithm for mobile environments should carefully balance these considerations as discussed in the following sections, an outline of which is given below.

Section 2 reviews existing work and evaluates techniques proposed by previous researchers. Section 3 presents a new consistency management algorithm, derived from previous work, which aims to provide greater efficiency within a mobile collaboration environment. Section 4 presents the findings of the simulation experiments conducted to measure the performance of the new algorithm and finally, Section 5 ends with a summary and conclusion.

2. Related Works

Consistency criteria can be classified into two classes, strong consistency and weak consistency [2]. Strong consistency prevents replicas from diverging by not allowing conflicting operations to be performed concurrently. This can be achieved by using locks [7, 10, 12] where a user has to obtain a lock before making a change on the shared object to ensure that no one else can make changes. However, these algorithms have several drawbacks. Firstly, they do not promote real-time concurrency since specific parts of the document can only be modified by one person at a time. Secondly, if a device acquires a lock on part of the document and then becomes unexpectedly disconnected (as commonly occurs in a mobile environment), no other site can access that part of the document for an indefinite period. Thirdly, the locking mechanism requires either a central server to manage the lock, or the additional overhead of message exchange to ensure that all sites agree on who acquires a lock on a particular part of the document at any given time. On the other hand, weak consistency allows divergence but might never enforce convergence. Therefore, this is not applicable in collaboration applications when the consistency of replicas is important.

Alternatively, another consistency class can be adopted where replicas are allowed to be temporarily divergent, and convergence is enforced by propagating the changes as soon as possible and by utilising an algorithm to ensure the conflicting operations are resolved in a consistent manner at all sites. This technique, called operation transformation, was first introduced by dOPT [1] and soon followed by other work such as adOPTed [9], GOT [15], GOTO [13], SOCT2 [11], SOCT3 [16], and SOCT4 [16]. Users are allowed to modify their respective local replicas concurrently with each update resulting in a local operation which is applied to the local replica before being broadcast as a *remote operation* to other participants. When a remote operation arrives at a certain site, that site performs an operation transformation to create a local operation variant which preserves the intention of the user who generated the original operation while resolving any conflicts and ensuring documents remain consistent across sites. This is done by transforming the incoming operation against all other operations that are concurrent to it. Lamport [5] defines causal precedence and concurrency as follows:

Definition 2.1. *Causal precedence relation "$\rightarrow$".*
Let op_i and op_j be operations generated at sites S_i and S_j respectively, op_i causally precedes op_j ($op_i \rightarrow op_j$) iff:

- *$S_i = S_j$ and op_i is generated before op_j, or*
- *$S_i \neq S_j$ and op_j is generated by site S_j after op_i is received by S_j.*

Definition 2.2. *Concurrent operations relation "$||$".*
Given two operations op_i and op_j, generated at sites S_i and S_j respectively, op_i and op_j are concurrent ($op_i||op_j$) if $op_i \not\rightarrow op_j$ and $op_j \not\rightarrow op_i$.

In the dOPT [1] algorithm, an incoming remote operation is transformed against its concurrent operations before being applied to the document and stored into the operation history. The major drawback of dOPT is that it naively transforms every remote operation against concurrent operations in the history without taking user intention into account. Guerraoui et al. [2] prove that dOPT is incapable of maintaining document consistency under some scenarios, particularly when operations are not generated at the same state. Ressel et al. [9] proposed the adOPTed algorithm to solve this problem by constructing an N-dimensional interaction graph (where N is the number of collaborating sites) that contains all operations in various possible

transformation variants. This approach, however, requires each site to construct a new graph every time a remote operation is received and thus as the number of concurrent operations and participating sites increases, so does the complexity of the graph. This makes it difficult to manage the graph over long collaboration sessions, particularly on resource constrained mobile devices. Alternatively, GOTO [13] and SOCT2 [11] try to solve this problem by ensuring that incoming remote operations are transformed against concurrent operations as if they were generated at the same state. This is done using a backward transformation technique as introduced by Prakash et al. [8], which separates the history into two sequences as shown in Figure 1. The first sequence consists of preceding operations, the second of concurrent operations, with the remote operation being transformed against the latter sequence. However, this leads to inconsistent states under some scenarios when concurrent operations are not performed in the same order at all sites [16].

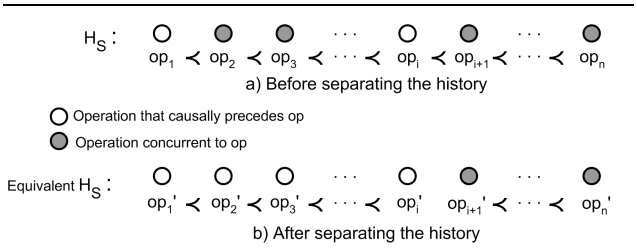

Figure 1. Separating the History.

GOT [15] and SOCT3 [16] implement a total ordering mechanism so that all operations are eventually executed in the same order at all sites. In GOT, when a remote operation is received, the operations that precede it are undone. The remote operation is then executed, followed by the execution of the operations that were previously undone after transforming them to include the effect of the newly executed remote operation. This algorithm is computationally expensive since it requires a large number of undo and redo operations (and their resulting transformations) and is thus not immediately suitable for use on mobile devices with limited processing power and battery life. In contrast, SOCT3 does not involve undo and redo operations, but rather, like SOCT2, when a remote operation is received, it is transformed against all concurrent operations followinghistory separation. Once the remote operation is executed, the history is rearranged such that the remote operation is positioned at its correct place in the history based on the total ordering scheme.

SOCT3 with its use of history separation, total ordering and operation shifting, serves as the most suitable basis for application in a mobile context. Nevertheless, it has a number of drawbacks. Firstly, it relies on a central sequencer for its total ordering mechanism. Secondly, it does not control history size and thus the longer a collaboration session runs, the more memory and/or storage space is consumed. Finally, the history separation step requires that the whole history be copied, with the separation performed on the copy so that the original history is left intact. This results in increased memory usage which, as discussed previously, is problematic on constrained mobile devices.

The first problem can be resolved by using an existing non-centralised total ordering strategy such as the State Vector (SV) technique as implemented by GOT [15], or the Lamport logical clock (LC) approach [5] as used by ORESTE [4]. The problem of history size can be addressed by implementing a history trimming algorithm that prevents the history from growing indefinitely. Such a technique has been introduced in a non-mobile context by Sun et al. [14]. However, since this algorithm can be applied independently at each site, it appears to be suitable for use in a mobile context. While these initial two problems can be potentially resolved using existing techniques, the problem of history copying has not been addressed and thus a novel technique called *Partial History Copy* is introduced to minimize the size of the history copy. These strategies are incorporated into a mobile collaboration algorithm which is presented and evaluated in this paper, with each specific technique discussed in more detail in the following section.

3. Algorithm Design Alternatives

As discussed in Section 2, there are three major issues with existing algorithms that should be addressed in a consistency management algorithm for real-time mobile applications. These are total ordering mechanism, history copy and history trimming.

3.1. Total Ordering Mechanism

As discussed previously the central sequencer approach used in SOCT3 and SOCT4 is not suitable for mobile applications because it introduces a single point of failure. Two existing candidate mechanisms for achieving total ordering, that do not rely on a central server or the synchronisation of clocks, are Lamport's Logical Clock as used by ORESTE [4] and the State Vector (SV) technique as implemented by GOT [15]. These two techniques are described below.

3.1.1. Lamport's Logical Clock Each operation is timestamped with a logical clock rather than a physical clock. Each site S_i maintains a logical clock C_i and whenever site S_i generates an operation op_j, op_j will be timestamped $C(op_j)$ where $C(op_j) = C_i$. C_i is incremented after assigning a clock to an operation to ensure the preceding operation has a lower logical clock than the following operation. A total ordering relation "$\prec$" of operations is defined as follows:

Definition 3.1. *Total ordering relation "$\prec$".*
Let op_i be generated by S_i and op_j by S_j, then $op_i \prec op_j$ if and only if:

- $C_i(op_i) < C_j(op_j)$, *or*
- $C_i(op_i) = C_j(op_j)$ *and* $S_i < S_j$.

3.1.2. State Vector A state vector, based on the clock vector introduced by Mattern [6], is an N-sized vector where N is the number of the participating sites. Each site S_i maintains a state vector V_{S_i}. $V_{S_i} = (V_{S_i}[1], V_{S_i}[2], \ldots, V_{S_i}[N])$ where $V_{S_i}[j]$ holds the number of operations generated by site S_j that have been executed by site S_i. If operation op_i is generated by site S_i, then op_i will bear a state vector V_{op_i}, which is equivalent to V_{S_i} right before S_i generates op_i. Sun et al. [15] defines a total ordering relation "$\prec$" of operations based on $sum(V_{op})$ as follows:

Definition 3.2. *Total ordering relation "$\prec$".*
Given two operations op_i and op_j, generated at sites S_i and S_j respectively, then $op_i \prec op_j$ iff:

- $sum(V_{op_i}) < sum(V_{op_j})$, *or*
- $sum(V_{op_i}) = sum(V_{op_j})$ *and* $S_i < S_j$.

The state vector technique has been used in the algorithm to preserve causality , therefore using the state vector for total ordering does not increase overall memory usage. However, since the total ordering relation involves a summation of the elements of the state vector, the computational intensity must be considered in a mobile context. On the other hand, using Lamport's logical clock as a total ordering mechanism will increase memory usage as each site and each operation need to maintain the logical clock. Given the potential advantages and disadvantages of the two total ordering schemes, we consider this in the empirical tests of Section 4 in order to determine which approach is the most suitable for use in a mobile context.

3.2. History Trimming

An operation op_i in the history is no longer required when there are no longer future operations that are concurrent to or precede it. This requires a site S_j to know that all other sites have already executed op_i before op_i can be deleted from the history of S_j. To do this, each site maintains information about the state vectors of all other sites. Let $SVT_i[j](1 \leq j \leq N)$ be the state vector of site S_j as known by site S_i, and $SVT_i[j][k]$ be the number of operations generated from site S_k that have been executed by site S_j as known by site S_i. To ensure $SVT_i[j]$ is up to date, whenever a remote operation op from site S_j is executed at site S_i, $SVT_i[j]$ is updated to be equal to V_{op}.

Let op be an operation generated from site S_k. Sites that have already executed op will have $V_S[k] \geq V_{op}[k]$, thus all operations that op precedes will have $V[k] \geq V_{op}[k]$. If site S_i receives an operation op_x from site S_y, site S_i will know that site S_y has already executed op if $V_{op_x}[k] \geq V_{op}[k]$. Therefore, if an operation op is generated from site S_k, op can be deleted from the history of S_i if $V_{op}[k] \leq SV_i[j][k](\forall j : 1 \leq j \leq N)$.

This history trimming technique requires additional memory to maintain SVT and more processing cycles to perform the trimming operation. However, over time, this technique is expected to both reduce memory usage and improve performance due to the smaller size of the history being operated on.

3.3. History Copy

Let H_{S_i} be the operation history of site S_i and $H_{S_i}[j]$ be the j-th operation in H_{S_i}. Operations in the history are totally ordered such that $H_{S_i}[j] \prec H_{S_i}[j+1]$. When a remote operation op arrives at S_i, the history separation step of SOCT3 arranges the history into two sequences, seq_1 and seq_2 ($H_{S_i} = seq_1 + seq_2$). All operations in seq_1 precede op and all operations in seq_2 are concurrent to op (Figure 1). This is done on the copy of the history so that the original order of the history is preserved. Since copying the entire history consumes both memory space and processing power, we propose a *partial history copy* technique that copies only the necessary portion of the history. The proposed technique aims to find an operation op_m in the history where all other operations located to the right of op_m ($\forall i : op_m \prec op_i$) are concurrent to op. If op_m is identified, then only operations to the left of it ($\forall i : op_i \prec op_m$) need to be copied and rearranged since they consist of operations that precede op and operations concurrent with op. The following Lemmas are introduced to help find the appropriate op_m.

Lemma 3.3. *If $op_i \rightarrow op_j$, then $op_i \prec op_j$.*

Proof. According to Definition 2.1, there are two possible cases where $op_i \rightarrow op_j$:

1. op_i and op_j are generated by the same site and op_i is generated before op_j. In this case $C(op_i) < C(op_j)$ if Lamport's clock is used for total ordering, and $sum(V_{op_i}) < sum(V_{op_j})$ if state vector is used for total ordering. Thus $op_i \prec op_j$ no matter what technique is used for total ordering.
2. op_i and op_j are generated by different sites and op_j is generated by site S_{op_j} after op_i is received by S_{op_j}. If Lamport's clock is used, S_{op_j} will update its logical clock C_j upon receiving op_i such that $C_j > C(op_i)$, thus $C(op_j) > C(op_i)$, which means $op_i \prec op_j$. If state vector is used, site S_j will only execute op_i if $V_{S_j}[k] \geq V_{op_i}[k](\forall k : 1 \leq k \leq N)$. Thus, after executing op_i, $sum(V_{S_j}) > sum(V_{op_i})$. op_j will bear state vector V_{op_j}, equal to V_{S_j}, which makes $sum(V_{op_j}) > sum(V_{op_i})$ and therefore $op_i \prec op_j$. Therefore $op_i \prec op_j$ no matter what technique is used for total ordering.

Therefore in either case (op_i and op_j are generated at the same site or not), if $op_i \rightarrow op_j$, then $op_i \prec op_j$. □

Lemma 3.4. *If $op_j \prec op_i$, then $op_i \not\rightarrow op_j$.*

Proof. The inverse of Lemma 3.3 is true that if $op_i \not\prec op_j$, then $op_i \not\rightarrow op_j$. Since $op_i \not\prec op_j$ is equivalent to $op_j \prec op_i$, the inverse can be restated as: if $op_j \prec op_i$, then $op_i \not\rightarrow op_j$. □

When a remote operation op arrives at site S_i, there is op_m in the history such that $op_m \prec op \prec op_{m+1}$. Since $op \prec op_{m+1}$, op_{m+1} and all other operations after it do not precede op (Lemma 3.4), they stay at their respective position in the history and only $[op_1 \dots op_m]$ needs to be rearranged and therefore copied (Figure 2). In other words, only operations that totally precede op ($(\forall i : op_i \prec op)$ need to be copied. The total ordering mechanism defined in Section 3.1 is used to determine the total precedence ($op_i \prec op$). The Partial Copy algorithm is outlined as follows:

```
Partial_Copy_History(H_S, op) {
    H'_S = [ ]; /* Initialize an empty history copy */
    j = 1;
    while(op_j ≺ op AND j ≤ N) {
        H'_S = H'_S + [op_j];
        j = j + 1;
    }
    return H'_S;
}
```

Although this technique is expected to minimise memory and processing usage over time, it consumes additional processing power when tracing the history to find the op_m. Therefore this technique is compared with the full history copy approach in the empirical study presented in the following section to determine whether the use of this algorithm is justified.

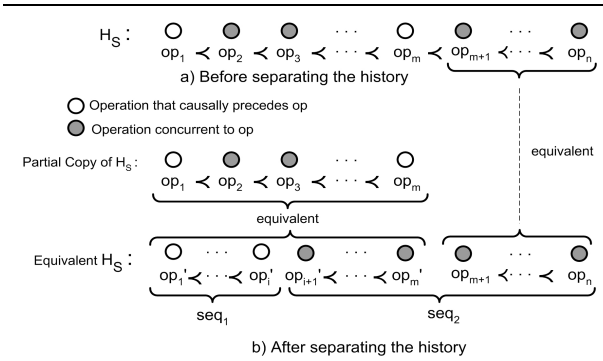

Figure 2. Separating the History Using Partial Copy.

The design factors and alternative implementations discussed in this section can be summarised as follows:

1. State Vector(SV) vs. Lamport's Clock(LC),
2. History Trimming(HT) vs. No History Trimming (NoHT), and
3. Full History Copy (FC) vs. Partial Copy (PC).

Based on those aspects, we can devise eight algorithm designs derived from combinations of the above techniques, which are compared empirically in the following section:

1. SV, NoHT, and FC
2. LC, NoHT, and FC
3. SV, HT, and FC
4. LC, HT, and FC
5. SV, NoHT, and PC
6. LC, NoHT, and PC
7. SV, HT, and PC
8. LC, HT, and PC

4. Experiments and Results

As discussed previously the aim of this paper is to propose and evaluate an efficient real-time consistency management algorithm for use in mobile collaboration environments. Based on the review of existing work and following on from the identification of, and solution to, a number of problems, this section presents an empirical study which compares a number of candidate algorithm variations in order to determine which one is most efficient in terms of performance and resource utilisation and thus most suitable for use in a mobile context. The experiments were based on simulations written in the Java programming language.

4.1. Independent Variables

Three independent variables are manipulated for each of the eight algorithm combinations identified in the previous section: *number of sites*, *number of operations* and *broadcast delay*. The higher the number of sites, the greater the number of remote operations and thus concurrent operations. The higher the number of generated operations at each site, the larger the history size. The impact of history trimming is expected to be more significant as the number of operations, and thus the history size, increases. The number of concurrent operations increases as this delay increases. The chosen broadcast delays are intended to be representative of realistic delays in a mobile environment, with the main intention is to investigate the trend in algorithm performance as delay increases.

4.2. Dependent Variables and Expected Outcomes

The efficiency characteristic of *performance* is operationally defined as the dependant variable *Execution Time* measured in seconds. The algorithm with the highest performance is the one with the lowest processing time, which relates to reduced power consumption and an enhanced user experience. The following are expected outcomes in terms of performance:

- **P-1**: LC will be faster than SV, since SV requires additional processing effort for the summation of the state vector elements.
- **P-2**: HT will be faster than NoHT. HT reduces history size, thus the remote operation process is expected to be faster.
- **P-3**: PC will be faster than FC. PC copies only part of the history and thus the time taken to perform history separation will be shorter.

The second efficiency characteristic of *resource utilisation* is operationally defined using two variables, *History Size after operation execution* and *History Copy Size during operation execution*, both of which relate to memory usage. The following are expected outcomes with regards to the resource utilisation:

- **M-1**: History size after operation execution is less for HT since HT regularly trims the history.
- **M-2**: PC uses less memory during a operation execution, since it does not copy the entire history.

Furthermore, the expected outcomes for the *performance* subcharacteristic (**P-1**, **P-2**, and **P-3**) also indirectly influence resource utilisation since the reduced processing overhead of an operation results in lower processor utilisation.

4.3. Results

Figure 3 shows the results in terms of performance, demonstrating that **P-2** is satisfied. The design alternatives involving history trimming (HT) perform better in terms of execution time than those without (NoHT). On average, HT reduces the execution time by almost 40% (Figure 3d) with the difference increasing as the number of sites and number of generated operations (and thus the total number of operations being exchanged) increases. The longer the collaboration runs, the greater the difference in history size between HT and NoHT (Figure 4a), thus the execution time difference between HT and NoHT also increases (Figure 3a and 3b). However, as the broadcast delay increases, the performance of HT gets closer to NoHT (Figure 3c). When the delay is 300ms, the difference is approximately 50% and when the delay is 8000ms, the difference is less than 20%. This is because the longer the broadcast delay, the less often the history gets trimmed. An operation in the history of a site can only be trimmed if that site knows that all other sites have already executed that operation, as derived from the state vector of the received operations (see Section 3.2). Consequently, when the network delay is high, this information arrives later, thus the history also gets trimmed later than when the network delay is low.

While **P-2** is strongly displayed by the graphs, expected outcome **P-1** does not hold since there are situations where SV is better than Lamport and vice versa. Therefore, based on execution time, there is no clear reason to favour SV or Lamport for total ordering. The same is true with **P-3** where PC does not improve the performance in terms of execution time. Although PC saves processing power by not copying the entire history, it involves additional condition checking while copying the history and thus does not improve processor usage overall. Therefore, to help determine which total ordering technique is better overall, the resource utilisation of each technique in terms of memory usage must be considered.

Figure 4a supports our prediction that without trimming, history size grows linearly towards infinity since each executed operation is stored in the history for the entire duration of the collaboration session. Depending upon the implementation, this may also impact storage requirements and increase processing overhead as parts of the history are paged to and from permanent storage. In contrast, the designs that implement History Trimming prevent this from happening, thus supporting **M-1** when either SV or Lamport is used for total ordering. Of particular interest is that history trim-

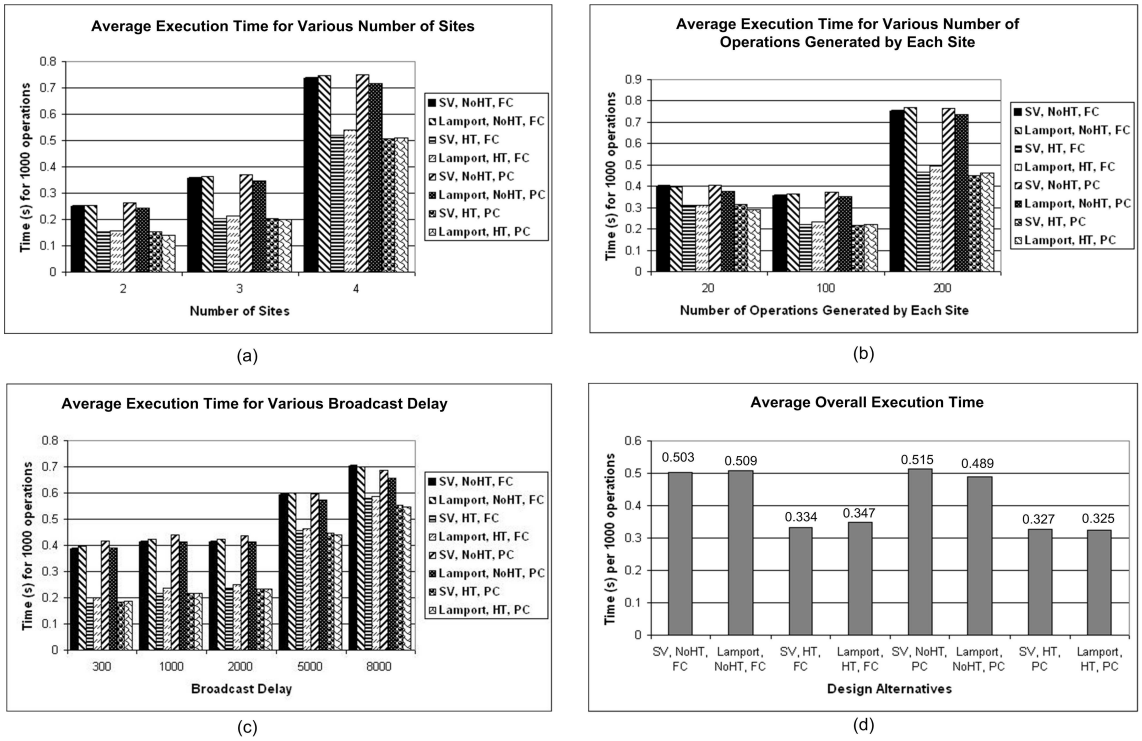

Figure 3. Simulation Results.

ming is more effective when SV is used in preference to Lamport. Therefore given that SV and Lamport exhibit similar performance characteristics in terms of execution time, SV is more efficient, and thus a better solution overall, since it results in less memory utilisation.

Figure 4b clearly supports the expectation **M-2** that PC reduces the size of the history copy and thus requires less memory to process remote operations regardless of whether or not HT is used. Therefore, since PC is neutral in terms of performance, as measured by execution time, due to its reduced memory utilisation it is superior in terms of efficiency to FC.

In summary, HT saves processing power and consumes less memory and is thus a clear choice in comparison to NoHT. For total ordering, SV is better than Lamport since even though they are similar in terms of performance, SV has lower memory utilisation requirements. PC is similar in that although not clearly superior in terms of performance it again reduces memory usage compared with FC. Therefore in summary, the algorithm variation that implements History Trimming and Partial History Copy, and uses State Vector for total ordering, is the most efficient and thus best choice for implementing a real-time collaboration algorithm in a mobile environment.

5. Conclusion

This paper has addressed the importance of efficiency as a software quality attribute, when designing a consistency management algorithm for use in a real-time mobile collaboration application. We propose and evaluate a new algorithm that is more efficient in terms of performance and resource utilisation and thus more suitable for use in a mobile context. The concepts of SOCT3 are used as a basis for the algorithm's correctness. The dependence of the algorithm on a central server is removed by incorporating known total ordering techniques, and the history trimming technique is added. Furthermore, the new algorithm introduced a novel partial history copying technique. The empirical testing indicates that a combination of History Trimming, Partial History Copy and State Vector for total ordering, produces the most efficient design for use in a mobile environment. Not only does this algorithm reduce overall execution time, it also reduces resource utilisation in terms of memory or fixed storage usage thus serving as a benchmark for comparison in any further research on this topic. While this paper has focused on efficiency, future work could involve the testing of additional ISO 9126-1 quality characteristics such as reliability and security. Such work would involve further use of simulation and possibly live test-

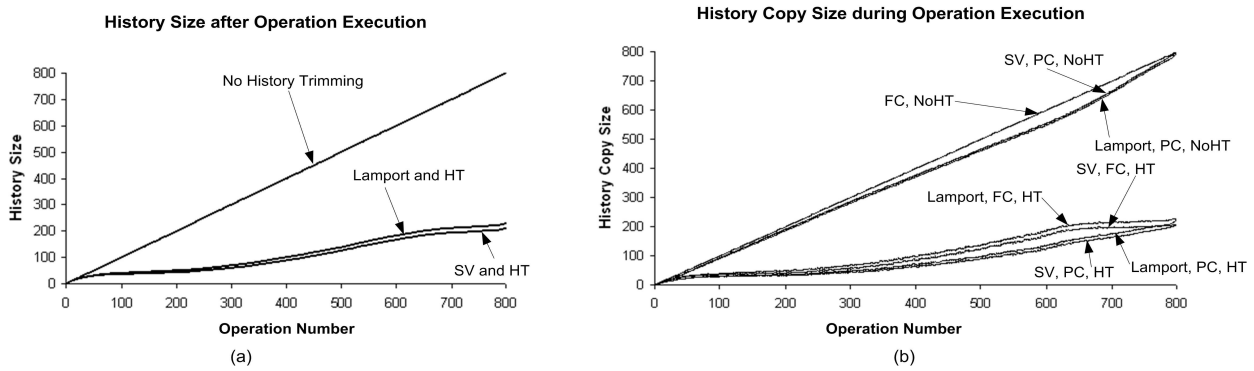

Figure 4. History Size.

ing using a real application, in addition to revised experimental designs based on the derivation and operational definition of a new set of variables to quantify the effects on the different quality attributes.

6. Acknowledgements

This work is part of the research program of the Australian Telecommunications Cooperative Research Centre (ATCRC) http://www.telecommunications.crc.org.au.

References

[1] C. A. Ellis and S. J. Gibbs. Concurrency control in groupware systems. In *Proceedings of the 1989 ACM SIGMOD international conference on Management of data*, pages 399–407. ACM Press, 1989.

[2] R. Guerraoui and C. Hari. On the consistency problem in mobile distributed computing. In *Proceedings of the second ACM international workshop on Principles of mobile computing*, pages 51–57. ACM Press, 2002.

[3] ISO/IEC. *FDIS 9126-1 Software Engineering - Product Quality - Part 1: Quality Model*. November 1999.

[4] A. Karsenty and M. Beaudouin-Lafon. An algorithm for distributed grouware applications. In *Proceedings the 13th International Conference on Distributed Computing Systems*, pages 195–202, May 1993.

[5] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Comunications of the ACM*, 21(7):558–565, July 1978.

[6] F. Mattern. Virtual time and global states of distributed systems. In *Proceedings of the International Workshop on Parallel and Distributed Algorithms*, pages 215–276. Elsevier Pub., 1989.

[7] J. Munson and P. Dewan. A concurrency control framework for collaborative systems. In *Proceedings of the 1996 ACM conference on Computer supported cooperative work*, pages 278–287. ACM Press, 1996.

[8] A. Prakash and M. J. Knister. Undoing actions in collaborative work. In *Proceedings of the 1992 ACM conference on Computer-supported cooperative work*, pages 273–280. ACM Press, 1992.

[9] M. Ressel, D. Nitsche-Ruhland, and R. Gunzenhäuser. An integrating, transformation-oriented approach to concurrency control and undo in group editors. In *Proceedings of the 1996 ACM conference on Computer supported cooperative work*, pages 288–297. ACM Press, 1996.

[10] M. Roseman and S. Greenberg. Building real-time groupware with GroupKit, a groupware toolkit. *ACM Trans. Comput.-Hum. Interact.*, 3(1):66–106, 1996.

[11] M. Suleiman, M. Cart, and J. Ferrié;. Serialization of concurrent operations in a distributed collaborative environment. In *Proceedings of the international ACM SIGGROUP conference on Supporting group work : the integration challenge*, pages 435–445. ACM Press, 1997.

[12] C. Sun. Optional and responsive fine-grain locking in internet-based collaborative systems. *IEEE Transactions on Parallel and Distributed Systems*, 13(9):994–1008, September 2002.

[13] C. Sun and C. Ellis. Operational transformation in real-time group editors: issues, algorithms, and achievements. In *Proceedings of the 1998 ACM conference on Computer supported cooperative work*, pages 59–68. ACM Press, 1998.

[14] C. Sun, X. Jia, Y. Zhang, Y. Yang, and D. Chen. Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems. *ACM Trans. Comput.-Hum. Interact.*, 5(1):63–108, 1998.

[15] C. Sun, Y. Zhang, X. Jia, and Y. Yang. A generic operation transformation scheme for consistency maintenance in real-time cooperative editing systems. In *Proceedings of the international ACM SIGGROUP conference on Supporting group work : the integration challenge*, pages 425–434. ACM Press, 1997.

[16] N. Vidot, M. Cart, J. Ferrié, and M. Suleiman. Copies convergence in a distributed real-time collaborative environment. In *Proceedings of the 2000 ACM conference on Computer supported cooperative work*, pages 171–180. ACM Press, 2000.