Electronic Communications of the EASST
Volume 17 (2009)

Workshops der Wissenschaftlichen Konferenz Kommunikation in Verteilten Systemen 2009 (WowKiVS 2009)

Towards Logical Clocks in P2P-based MMVEs

Torben Weis, Arno Wacker, Sebastian Schuster, Sebastian Holzapfel

9 pages

Guest Editors: M. Wagner, D. Hogrefe, K. Geihs, K. David
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Towards Logical Clocks in P2P-based MMVEs

Torben Weis, Arno Wacker, Sebastian Schuster, Sebastian Holzapfel

University of Duisburg-Essen

Abstract: A crucial requirement for peer-to-peer-based Massively Multiuser Virtual Environments (P2P-based MMVEs) is the accurate and reliable synchronization of actions among the users (processes). To do so, clock synchronization protocols can be used. In this paper we first analyze the usage of standard vector clocks for this purpose and show their deficits (e.g. growing large). Then we present a novel variation of vector clocks – *pruned vector clocks* – which overcome the deficits of standard vector clocks and are therefore suited for their usage in MMVEs. The basic idea of pruned vector clocks is to prune all entries in a vector clock, which are not relevant at some point in time. We show, that with this approach vector clocks will stay constant in size and still provide the necessarily synchronization among the processes.

Keywords: MMVE, Synchronization, Clocks, Consistency

1 Introduction

In this paper we will present a variation of vector clocks to synchronize actions in P2P-based MMVEs. A MMVE is a large distributed system where thousands of users cooperatively enjoy a virtual world. In this world they can perform actions, e.g. trade, talk, or engage themselves in a virtual battle. A basic algorithmic problem in such cooperative distributed systems is synchronization by means of clocks. Physical clocks are always problematic due to their inherent time drift. In other cooperative distributed applications such as cooperative text editing logical clocks are the best choice. They do not suffer from drifting clocks and they can yield the causal dependencies of actions carried out at different computers across the network.

The most popular MMVEs today are used for game play, e.g. World of Warcraft [Bli05] or Eve Online [CCP03]. These systems rely on a very huge and expensive server infrastructure. However, this simplifies the clock problem. All players send their actions to the server which decides the order of the actions and reports that back to the players' computers. Thus, at least in theory all players see the actions in the same order.

In our paper we assume a P2P infrastructure, i.e. there is no central server which can coordinate all actions happening in the world. The advantage of a P2P infrastructure is that it can potentially scale with the number of users in the system. On the other hand it opens new algorithmic challenges, for example clocks are more difficult to handle. We will present a variation of vector clocks which can handle MMVEs with many thousand users online. Traditional vector clocks are not well suited since they would simply grow too large, i.e. one entry for every user. Our approach will be based on sparse vector clocks.

This paper is structured as follows. In the next section we are discussing a seemingly obvious approach to handle vector clocks in environments with many thousand cooperative users and

discuss it's deficiencies. In the following section we present a better solution called "pruned vector clocks". Then we discuss in Section 4 related work and close this paper with conclusions and outlook in Section 5.

2 Sparse Vector Clocks

Vector clocks are linearly growing with the number of participants in the distributed system. Since one MMVE can host many thousand users at a time, this would easily lead to a situation where the vector clock won't fit in a single UDP datagram and will therefore make it very expensive to notify other clients about an action. In an ideal case one unfragmented UDP datagram should be sufficient. Otherwise the network load is increased and we need to handle reordering and packet drops.

Thus, it is a strict requirement to reduce the size of the vector. In this chapter we will present sparse vector clocks which are an seemingly obvious modification of traditional vector clocks. However, at the end of the section we will show the limitations of this first approach.

In an ideal case the size of the vector clock does not depend on the total number of users in the MMVE. It should only depend on the set of users which are important to a certain action. In virtual combats all players engaged in the battle are important and therefore their components cannot be removed from the vector clock. However, players which are not directly engaged in the battle or are far away in the virtual world are not important. Their components could be removed from the vector. This leads to a definition of sparse vector clocks.

Definition 1 (Sparse Vector Clock) A vector clock $V = (v_1, v_2, \cdots, v_n)$ is called sparse with respect to a constant $s \leq n$ if the number of entries of $v_i > 0$ is smaller or equal to s. Thus, in a sparse vector clock a certain number of entries are zero.

This allows for an obvious compression of the vector. The following is an example of a sparse vector:

$$V_{Sparse} = (0,0,0,0,0,123,0,0,0,25,0,0,0,0,345,0,0,0,0)$$

The vector does not contain the user IDs. It is assumed that all users are sorted from 1 to n and therefore the position v_i in a vector clock V is the clock component of the i-th user. In MMVEs no user has a complete list of all other users. Thus, the concept of "user number i" cannot be applied. Instead, we encode the sparse vector clock from above as a dictionary as follows:

$$V_{Sparse} = \{UserID5960 : 123, UserID4353 : 25, UserID9879 : 345\}$$

Thus, the dictionary-based vector clocks can be treated as a set of key value pairs.

Definition 2 (Dictionary-based Vector Clock) A dictionary-based vector clock V is a subset of the space $ID \times N$, where ID is the space of all possible user IDs and N are the natural numbers. No two tuples (id, c_1) and (id, c_2) can coexist in this subset. The clock component of a user X can be obtained by searching for a tuple (X, c) in the subset. If such a tuple exists, the clock component is c, otherwise the clock component is by definition 0.

Sparse vector clocks can be compared like normal vector clocks. In the end, it is just a different encoding but mathematically not different from standard vector clocks.

2.1 Essential, Related and Unrelated Clocks

As stated above, an MMVE implementation must define means to distinguish important from unimportant actions. Instead of deciding this on a per-action basis, we identify players who are not related to the play of some player A. The clock components of these unrelated players would simply be set to zero in the vector clocks of player A.

A first rule of thumb for identifying unrelated players is that for a player A all players whom he has never met in the virtual world are inherently uninteresting. Furthermore, players out of sight can be neglected, too, since they are not important for the synchronization of events which they can neither see nor influence due to their geographical distance. We call the clock components of these players unrelated since the actions of player A are not related to their actions.

In a scalable P2P-based MMVE the system will report actions only to those players who can either see them or who are directly influenced. Thus, dropping the above mentioned unrelated clock components will not change the behavior of the MMVE, but it can reduce bandwidth.

Sometimes even more bandwidth must be saved by transmitting even less vector clock components. Therefore, we must identify essential players whose clock components must not be dropped. These clock components are called essential. For example, if two players talk, trade, or hit each other, we cannot simply drop one action. All participating players must agree on the outcome of the trade and in combat all combatants must agree which player made the final (and therefore lethal) strike since this is rewarded with bonus points.

The clock components of all actions which are only observed by a player are not essential. Thus if players A and B are engaged in a fight and players C and D are engaged in a different fight then player A is only an observer of the actions between C and D. In this case it is acceptable if these two fights are not synchronized with each other. Hence, players A and C can for example not agree which fight terminated first. Due to the sparse vector clocks, player A cannot relate the actions of player C to his own actions. However, the sequence of actions between C and D will be observed correctly by player A. To relate these two independent fights to each other, we will revert to wallclock time.

2.2 Shortcomings of Sparse Vector Clocks

Unfortunately, sparse vector clocks have two remaining problems. The first problem is related to the clock update. If player A gets a vector clock from player B, he builds the supremum of both vectors to update his clock. For example if the vector clock of player A looks like this

$$V_A = \{A : 123, B : 345, F : 125, Q : 12\}$$

and the vector clock of player B is as follows

$$V_B = \{A : 123, B : 346, F : 126, P : 64\}$$

then the supremum will yield

$$V'_A = sup(V_A, V_B) = \{A : 123, B : 346, F : 126, P : 64, Q : 12\}.$$

This shows that the vector clock is constantly growing. The more players one meets, the larger the vector clock will become. In real world examples research showed that due to the "small word" phenomenon every person knows every other person over very few links, because some persons act as hubs, i.e. they know many other persons. The same may happen in an MMVE. Some players are very active and have many friends. Meeting such a "hub player" will cause a significant increase of the vector clocks. Quickly the sparse vector clock will no longer be sparse.

3 Pruned Vector Clocks

In the previous section we could show that sparse vector clocks have the problem of losing their sparse property due to the way the supremum is built. Pruned vector clocks modify the supremum construction and the algorithm for comparing two arbitrary pruned vector clocks.

3.1 Building the Supremum

A pruned vector is represented as a dictionary (see Definition 2) and gets its name from a final step being applied after the supremum is calculated. If player A with vector clock V_A receives a vector clock V_B of player B, he builds the supremum $V'_A = sup(V_A, V_B)$ as follows:

Algorithm 1 Player A building supremum of V_A and V_B, i.e. $V'_A = sup(V_A, V_B)$

1: **for all** $(id,t) \in V_B$ **do**
2: **if** $\exists (id,x) \in V_A$ **then**
3: $V'_A = (V_A - \{(id,x)\}) \cup \{(id, max(t,x))\}$; // There is a local clock component for this id, hence update.
4: **else**
5: $V'_A = V_A \cup \{(id,t)\}$; // No local clock component information about id, hence add.
6: **end if**
7: **end for**
8: // Now prune the vector
9: **for all** $(id,y) \in V_A$ **do**
10: **if** $\neg\, isRelevant_A(id)$ **then**
11: $V_A = V_A - \{(id,y)\}$;
12: **end if**
13: **end for**

The difference to sparse vector clocks is the use of the `isRelevant()` function. Vector clock components of users who are not relevant to A are pruned. This avoids that the memory footprint of the vector clock increases over time. If the function `isRelevant()` yields true for a constant number of players, we can guarantee that the vector V_A always retains its sparse property if it was sparse before the supremum building. When player A enters the world his vector clock has only one entry for himself, thus it is sparse. It follows that it will always be sparse.

3.2 Comparing Pruned Vector Clocks

However, we must modify the way two pruned vector clocks are compared, too. To illustrate the necessity of a modification we provide in the following an example where two players are unable to determine any order of their actions. In Figure 1 four players are depicted. The arrows show which players are directly interacting. The dotted ellipse illustrates the viewing range of players A and B. Thus, player A cannot see X and player B cannot see Z.

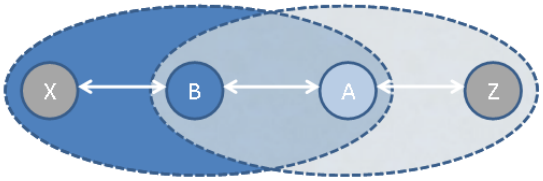

Figure 1: Example of interacting players

If player A receives a vector clock from player B he will prune the vector clock component of player X since X is not relevant to A. The same happens to the vector clock component of Z when B prunes its vector clock. Thus, the clocks of A and B will always have the form $V_A = \{A : a, B : b, Z : z\}$ and $V_B = \{B : b, A : a, X : x\}$ for some numbers $a, b, c, x, z \in N$.

Whenever we directly compare the vectors V_A and V_B we will get the result that they denote two actions which happened in parallel although this is not the case. The following example proofs the claim. Assume that players interact sequentially in the following order: $X \rightarrow B$, $B \rightarrow A$, $A \rightarrow Z$, $Z \leftarrow A$, $A \rightarrow B$. It follows that the last action $(A \rightarrow B)$ is indeed later than the second action $(B \rightarrow A)$ and the vector clock should reflect this. The following sequence of exchanged vector clocks shows that this is not the case.

1. Initially all clocks are empty
2. X updates its clock to $\{X : 1\}$ and sends it to B
3. B performs an action and updates its clock to $\{X : 1, B : 1\}$ and sends it to A
4. A performs an action and updates/prunes its clock to $\{B : 1, A : 1\}$ and sends it to Z
5. Z performs an action and updates/prunes its clock to $\{A : 1, Z : 1\}$ and sends it to A
6. A performs an action and updates its clock to $\{B : 1, A : 2, Z : 1\}$ and sends it to B
7. B compares the clock it got from A and its own clock from step 2

In step 7 it becomes obvious that the two vectors $V2 = \{X : 1, B : 1\}$ and $V7 = \{B : 1, A : 2, Z : 1\}$ seem to denote actions which happened in parallel, because $V2$ has a X clock component and $V7$ has a Z component, thus neither $V2 \leq V7$ nor $V2 \geq V7$. This problem happened because of the pruning while building the supremum. The solution is to identify the common subset of clock components and to prune the other ones. After the pruning, both vectors can be compared.

This yields the pruned vectors $V2' = \{B:1\}$ and $V7' = \{B:1, A:2\}$. When comparing the two vectors the result is as intended: $V2' < V7'$.

The following algorithm compares two pruned vector clocks. Without loss of generality we show how to implement the $<$ relation. All other relations ($>$,$||$,$=$,$<=$,$>=$) can be implemented accordingly.

Algorithm 2 Player A comparing two pruned vector clocks V_A and V_B, i.e. $V_A < V_B$

```
1:  V'_A = {}
2:  V'_B = {}
3:  for all (id,t) in V_A do
4:      if isRelevant_A(id) then
5:          V'_A = V'_A ∪ {(id,t)}
6:      end if
7:  end for
8:  for all (id,t) in V_B do
9:      if isRelevant_A(id) then
10:         V'_B = V'_B ∪ {(id,t)}
11:     end if
12: end for
13: return V'_A <_S V'_B
```

In the above algorithm we used the symbol $<_S$ to denote sparse vector clock comparison whereas the $<$ symbol denotes the comparison of two pruned vector clocks. The algorithm prunes the two vectors V_A and V_B yielding V'_A and V'_B by keeping only those vector clock components which are relevant to player A. Then the pruned vectors are compared like sparse vector clocks.

The comparison of two vector clocks can yield different results for different players. This would of course not happen with normal vector clocks. However, with pruned vector clocks we accept that different players see the actions in different orders. By using the `isRelevant()` function, we can at least guarantee that essential actions are seen in the correct order while we revert to unprecise wallclock time for non essential actions.

3.3 Restrictions

Pruned vector clocks exhibit a reduced memory footprint compared to standard vector clocks when used in an MMVE as described above. However, this reduction in size comes at a cost. Some of the expressiveness is lost. But we will show that this lack of expressiveness is no real restriction to MMVEs.

If two normal vector clocks $V1$ and $V2$ are in the relation $V1 < V2$ then the pruning the vector clocks to $V1'$ and $V2'$ will still yield $V1' < V2'$. This applies to the relations $<, >, \leq, \geq$ and $=$. The proof is simple. If all components of two vectors are pairwise in some relation (e.g. $V1[x] < V2[x]$ for all $1 \leq x \leq |V1|$) then any subset of these components is pairwise in the same relation.

The case is different for the parallel relation ("$||$" in short). If two normal vectors $V1$ and $V2$

are in the relation $V1||V2$ then the two pruned vectors $V1'$ and $V2'$ can be in any other relation. The reason is that the components of $V1$ and $V2$ are pairwise in different relations. Depending on which subset of vector components survives the pruning, some relations may no longer appear.

However, this is not a severe restriction. Vector clocks offer a partial order and two vector clocks which are parallel are usually forced in a $V1 < V2$ relation by some heuristics to achieve a total ordering. Pruned vector clocks sometimes do just that: they force two vector clock times into a "is-earlier-than" relationship although logically both vector clock times are parallel.

4 Related Work

Global ordering of events in a distributed system by physical timestamps is impossible because clocks are not perfectly synchronized. Lamport introduced the notion of a partial order of events in [Lam78], introducing the happen-before-relation to capture the causal relationship of events. He also introduced logical clocks yielding a total order of events, also ordering events that happened concurrently according to the causal order. Vector clocks introduced by Mattern [Mat89] and Fidge [Fid88] realize an order isomorphic to causal order.

As the size of the vector grows with the number of processes in the system, multiple proposals have been made to reduce memory needs and size of exchanged messages. Singhal and Kshemkalyani [SK92] introduced an algorithm to only exchange differential information between processes, trading local memory for bandwidths savings. To make vector clocks suitable for dynamic systems with processes joining and leaving the system, proposals to prune vector clocks have been made by [TA96, Sai02] and [Ric98] for example. Our proposal is comparable to these approaches but our decision to prune is based on spatial relationships in the virtual world and not on global consensus [Ric98] or loosely synchronized clocks [Sai02]. Charron-Bost has shown that for every approach reducing the vector size, scenarios can be designed where this approach fails to capture concurrent events according to the causal order [CB91]. However, as discussed in Section 3.3 this restriction is acceptable in our application settings.

In the area of distributed virtual environments, multiple proposals to capture the order of events and ensure consistency have been made. [ZCTL02] argued that causal order captures all possible causal relations, no matter which events are truly causally related. They introduced the notion of critical causality to capture the real cause of events. However, their definition is based on the assumption that only events that directly precede each other are causally related. This is not always true. In the end, only the user or the application itself knows about true causal relations. Our approach is somewhat similar, using the application knowledge about spatial relations to exclude causal relations between events that are too far away from each other. As distributed virtual environments are inherently realtime simulations, using only logical time for deciding when to execute events is not sufficient. Sufficient Causal Ordering [RS95] combined timestamps of wallclock time with logical timestamps. In our approach we fall back to this approach if the pruned vector clocks are parallel.

5 Conclusions

We presented a logical clock for MMVEs that is based on vector clocks. Unlike vector clocks we do not need to know all participants of the MMVE and the memory footprint of each player's clock can be constant. To achieve this we had to prune many components from the vector clock. To achieve this we presented several approaches for identifying candidates for pruning.

We introduced sparse vector clocks and argued that they will grow over time and will eventually converge against a complete vector clock thus losing the sparse property. Therefore, we introduced pruned vector clocks which work on the same clock representation but require different algorithms for updating and comparing clocks. We presented an algorithm for building the supremum of two pruned vector clocks and we presented an algorithm for comparing them.

We compared the expressive power of pruned vector clocks with that of traditional vector clocks. We could show that pruned vector clocks sometimes introduce an "is-earlier-than" relationship where the normal vector clock would have stated that the two vector times are parallel. We argued that this is no limitation in the settings of an MMVE.

Thus, we could show that pruned vector clocks are a practical solution for MMVEs with a reduced memory footprint and no substantial restrictions.

Acknowledgment

The work presented in this paper has partially been financed by the DFG Emmy Noether program.

Bibliography

[Bli05] Blizzard Entertainment Inc. World of Warcraft. http://www.worldofwarcraft.com, 2005.

[CCP03] CCP hf. Eve Online. http://www.eve-online.com, 2003.

[CB91] B. Charron-Bost. Concerning the size of logical clocks in distributed systems. *Information Processing Letters* 39(1):11–16, July 1991.

[Fid88] C. J. Fidge. Timestamps in Message-Passing Systems that Preserve the Partial Ordering. In *Proc. 11th Australian Comp. Sci. Conf.* Pp. 56–66. 1988.

[Lam78] L. Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM* 21(7):558–565, July 1978.

[Mat89] F. Mattern. Virtual Time and Global States of Distributed Systems. In Cosnard et al. (eds.), *International Workshop on Parallel and Distributed Algorithms*. Pp. 215–226. Elsevier Science Publishers, Amsterdam, 1989.

[Ric98] G. G. Richard III. Efficient Vector Time with Dynamic Process Creation and Termination. *Journal of Parallel and Distributed Computing (JPDC)* 55(1):109–120, Nov. 25 1998.

[RS95] P. Roberts, D.J.and Sharkey, P. Sandoz. A Real-time, Predictive Architecture for Distributed Virtual Reality. In *Proceedings of the 1st ACM SIGGRAPH Workshop Simulation & Interaction in Virtual Environments*. Pp. 279–288. Des Monies, Iowa, July 1995.

[Sai02] Y. Saito. Unilateral Version Vector Pruning using Loosely Synchronized Clocks. Technical report HPL-2002-51, Hewlett Packard Laboratories, Mar. 15 2002. http://www.hpl.hp.com/techreports/2002/HPL-2002-51.pdf

[SK92] M. Singhal, A. Kshemkalyani. An efficient implementation of vector clocks. *Inf. Process. Lett.* 43:47–52, 1992.

[TA96] F. J. Torres-Rojas, M. Ahamad. Plausible Clocks: Constant Size Logical Clocks for Distributed Systems. In Babaoglu and Marzullo (eds.), *Distributed Algorithms, 10th International Workshop, WDAG '96*. Lecture Notes in Computer Science 1151, pp. 71–88. Springer, Bologna, Italy, 9–11 Oct. 1996.

[ZCTL02] S. Zhou, W. Cai, S. J. Turner, F. B. S. Lee. Critical causality in distributed virtual environments. In *Proceedings of the 16th Workshop on Parallel and Distributed Simulation (16th PADS'02)*. Pp. 53–59. ACM, Washington, DC, USA, May 2002.