Common Notion of Time in the MIDAS MANET: Arrogant Clocks

Joe Gorman, Leendert W. M. Wienhofen
Joe.Gorman@sintef.no, Leendert.Wienhofen@sintef.no,
Software Engineering, Safety and Security, SINTEF ICT
Trondheim, Norway

Abstract

The EU-IST-MIDAS project has developed middleware that helps to simplify development of software for mobile ad-hoc networks (MANET) with distributed data. One of the challenges in distributed systems is the time difference in the various nodes that are part of the MANET. Time differences can disrupt the consistency of the data stored in a distributed database upon replication. This paper argues why clock synchronisation does not solve the needs of the MIDAS MANET, and introduces the concept of arrogant clocks as a method for maintaining a common notion of time in order to improve consistency when replicating database records between the different nodes in the MANET.

Introduction

MIDAS is a research project in the FP6 IST Programme of the European Commission. Its aim is to simplify the development of mobile applications for multiple, co-operating users who make use of a mixture of infrastructure-based communications and ad-hoc communications. An example of such an application would be to provide support for emergency workers responding to a major accident. Workers at the scene of the accident would be able to enter and access data using mobile terminals, and this information would be made available to others at the scene, to coordinating staff at other locations, and to medical teams at hospitals etc.

To be able to support applications of this kind, MIDAS assumes that communications infrastructure may not be available at all locations, necessitating the use of MANETs (mobile ad-hoc networks) by some users. Middleware developed in MIDAS integrates MANETs with infrastructure-based networks, but the communications links used for this will in general not be 100% reliable. However, application instances running on individual nodes must be able to run 100% of the time, with communications disruptions being handled in a way that does not interfere with user operations. A robust and autonomous approach is required that can cope with network partitioning i.e. the existence of multiple network “islands” each containing one or more communicating nodes, but not being able to communicate with the other “islands”. Because network partitioning can occur, access to a central server cannot be assumed.

The project has implemented middleware providing core functionality to support development of such applications. One of the key components of the middleware is the MDS (MIDAS data space), which provides a mechanism to share data between nodes by providing standard functions which ensure that data entered on any node is made available on all other nodes. This is achieved by using a mixture of remote accesses to other nodes and creating replicas of entire tables on multiple nodes. This is done in a transparent manner: application instances running on the different nodes do not need to know where replicas are stored. The MDS automates synchronisation of all replicas of all tables.

In the type of applications envisaged in MIDAS, a lot of the data will consist of observations about real-world entities, and these observations will be entered by different users, on different nodes, at different times. In an emergency support

This paper was presented at the NIK 2008 conference. For more information see http://www.nik.no/

application, for instance, paramedics may wish to record key medical indicators about the status of casualties at the scene of the accident (pulse rate, level of consciousness etc.), together with an assessment of evacuation priority (usually on simple 1-3 scale). For such observational data, it will be natural to add timestamp information to the basic observational data, as the time at which an observation was made may be directly relevant to decisions taken on the basis of that data by other people involved at later stages of the process. In addition, timestamps may be required due to formal requirements to produce a comprehensive official log. A third reason for needing timestamps is that they can provide a useful mechanism to help achieve data consistency.

In order for timestamps to be useful in practice, the different nodes in the network need to have some common notion of time. This means that – for any pair of observations – all nodes should "agree" on whether the two observations took place at the same time, or whether one was X minutes before the other etc. But it is not a simple matter to provide a common notion of time, given that MIDAS has no central server that could provide an "authoritative" notion of time. In the middleware and test applications developed in the project, a common notion of time is provided by making sure that clocks on all participating nodes are manually synchronised to some standard time prior to application deployment. This is impractical and inconvenient, and not feasible in all application scenarios. Thus, some other way of achieving a common notion of time is needed, and this paper presents a proposed way of doing so. Note that this approach has not yet been implemented in the MIDAS middleware.

Shared dataspaces for mobile networks: related work

Earlier work with aims similar to MIDAS in creating a shared dataspace was carried out in the Bayou project [3]. However, that approach was based on a set of servers. In MIDAS we aim to provide a solution which is independent of servers, in order to allow autonomous operation in network partitions (with full synchronisation achieved after partitions are re-united). An approach to this was developed and implemented by project partner SINTEF in the FieldCare project [1]. The FieldCare approach offers fully reliable synchronisation in ad-hoc networks, without any server. However, it is based on a distribution policy which replicates *all* data to *all* nodes. While this offers maximal data availability, it is not practical in terms of use of bandwidth and storage capacity, and therefore MIDAS aims to provide a more practical approach.

Project partner University of Oslo has earlier carried out extensive work on the problem ([5],[6],[7],[8],[9]), and developed an overall architecture identifying the detailed components needed to provide a hybrid solution. This work has been a major influence on the overall design of the MIDAS MDS component – but it did not address the problem of achieving a common notion of time in such a system, or the associated issue of achieving data consistency.

Data Consistency in MIDAS

Definition and ambition level

In a distributed database system, full data consistency means that the content of all tables on all nodes is identical at all times. This means that for any given table and primary key value *x*, a query issued at time *t* specifying *x* as primary key will return the same value for non-key fields, regardless of the node on which it is executed. For instance, in an emergency application, if a query about the evacuation priority of

casualty *Jim* (primary key) returns the value 1 on node 4 at time 14:07:15, then this same value would be returned for *Jim* on all other nodes at that time. In standard database systems, it is usually the aim to provide such full consistency. To do this, a *pessimistic* replication strategy must be followed, in which protocols ensure that a transaction is not committed until the update has been achieved on all nodes. However, as explained in [11], such a strategy cannot be adopted in MIDAS because:

- Some network connections may be unavailable for long periods, isolating some nodes. There could therefore be a very considerable delay before a transaction could be committed; this would mean that the update would not be available *anywhere* for a long time – which conflicts with requirements from the scenarios we are looking at.
- The network topology is dynamic, and new nodes can join at any time. New nodes may also be included in the set of nodes to which a given table is to be replicated. Thus, the set of nodes to which a transaction must be replicated is dynamic – and cannot be predicted in advance. For as long as new nodes keep joining the network, the transaction could not be committed – and that could be a very long time indeed.

Because pessimistic replication is not appropriate, in MIDAS synchronization is achieved using *optimistic replication.* This approach, described in detail in [4], essentially involves using a "best effort" approach to propagate changes to other nodes. Using such a strategy, a query about the evacuation priority for *Jim* might return the value 1 on one node – but respond with "No such patient as Jim" on another node. Application programmers must accept that there can be periods during which an update is available on some nodes, but not on others.

But can we be sure that – when a record for a given primary key has first been created on a node – that the same non-key values will be returned? The answer in the general case in MIDAS has to be "no". This is because the form of optimistic replication supported in MIDAS allows for multiple writers. Thus, for a given primary key value in a MIDAS table, it may happen that two independent application instances on different nodes update the entry for the same key to different values. For instance: in an application for emergency crews there might be a table showing the current evacuation priority of each casualty. Different medics attending the casualty might – independently, at different times – have entered different priority values, depending on the current status of the casualty. As there is, in general, no guarantee that there will always be connectivity between different nodes, the MDS must allow for the possibility that these independent writers enter values for the same casualty. When these nodes later establish connectivity with other parts of the network, two different values for Jim's evacuation priority will then be stored in the MDS tables.

The disadvantages of this potential for inconsistency must be weighed against the disadvantages of suffering very long delays in data becoming available at all. The view taken by MIDAS is that availability is more important than complete correctness, in the types of scenarios we wish to address. However, some data might be highly critical to users of the application, and in such cases correctness becomes more important. Evacuation priority in an emergency scenario is a good example. It would not be good if an application instance on a particular node returns priority 3 (i.e. the casualty does not need to go to hospital right away) if another medic later realised that the patient's condition had worsened and increased priority to 1 (i.e. immediate evacuation), but this data was not available. In some cases this is simply unavoidable: if the node used by the medic increasing priority to 1 never establishes contact with other nodes, then the data

is simply not available. But if the data *does* get synchronised with other nodes, then it becomes crucial to know which value was entered first: priority 3 or priority 1.

In connection with such concerns, we need to consider the type of timescales we expect in the type of applications for which MIDAS is intended. It is *not* our ambition to be able to support highly time-critical applications, where the order of real-world events may be critical at the level of seconds or even less. Rather, we make the assumption that criticality will only arise at timescales in the order of minutes or even more. Relating this to the evacuation priority example: we assume that, if different medics do make different decisions about the casualty's evacuation priority, then at least a few minutes will have gone by between the two assessments.

To summarise: our ambition level for data consistency in MIDAS is as follows:

- Availability is of greater importance than full data consistency; end-users of applications developed by MIDAS must be aware that it may sometimes happen that out-of-date data is displayed on a user terminal.
- MIDAS internal mechanisms to avoid inconsistencies will be considered good enough if they manage to determine the correct order of observations that took place within *minutes* of each other; conflict resolution at the level of seconds is not required.

MIDAS approach to Data Consistency

The core of the MIDAS approach to data consistency is to allow *multiple values* for a given primary key in a table. This contrasts fundamentally with normal database behaviour, where it is not possible to store more than one value for the same primary key. It is motivated by the fact that multiple writers may update the same record. This can be regarded as a feature rather than a limitation: it provides a facility for "versioning" of data. This can be directly useful in some applications e.g. where it is of interest to record the historical values of a piece of data (such as the pulse rate of a casualty over time)

However, allowing multiple values requires that some mechanisms be provided for conflict resolution (i.e. selection of which of the multiple values is the "correct" one). MIDAS provides two approaches to this:

- To allow use of what [4] refers to as *semantic* scheduling. This means that the application must resolve conflicts using application/domain-specific knowledge. In an emergency application, for example, the rule could be to always select the highest evacuation priority – to be on the safe side.
- To support use of what [4] refers to as *syntactic* conflict resolution i.e. automated resolution based on information available within the middleware itself. The chosen method for this is to base conflict resolution on timestamps (i.e. always select the "latest" record) – based on some system-wide common notion of time.

Inappropriateness of clock synchronisation in the MIDAS context

The time problem in distributed systems has been known and researched on for a long time [12] [13] [14]. Work has focused on mechanisms to synchronise clocks, to provide a common form of time stamping. Indeed, we have noticed that when describing MIDAS scenarios to people, the immediate reaction is always "you need to find a way to synchronise the clocks". The Network Time Protocol [10] is often suggested, as are other approaches to finding global states in distributed systems - for

example Lamport clocks, Logical clocks and Vector Clocks, as described in the works of Lamport [12], Chandy and Lamport [13] and Mattern [14].

However, clock synchronisation is not suitable for the scenarios envisaged in MIDAS. This is because independent network partitions may exist, and there may *never* be connectivity to any central server with an authoritative clock. It could happen that the nodes within one partition would successfully synchronise their clocks to one value, but that nodes in a separate partition would synchronise to another value. If these partitions were united at a later stage there would be an inconsistency in the timestamps for all the information shared within the partitions. A re-synchronisation of all the clocks could be carried out, but the old timestamp values would then be out-of-date, and need to be updated. This process would in itself take time, adding further complexity to the task - as the possibility of a new partitioning of the network would also need to be taken into account.

It is also important to remember that clock synchronisation takes time – and that during that time conflict resolution will not work properly.

At first glance, vector clocks seem an appropriate solution to resolve the ordering problem. Vector clocks keep an internal logical clock (initialised at 0) which increments its vector by one each time a local process is carried out. When a process receives a message from another node, the highest vector for that node is picked. So, when process A (with the status A=1, B=0, C=1) receives a message from B (A=0, B=2, C=1), it will only update the value for B to 1 as it is higher than the current record, equal and lower values are disregarded. For a more detailed description, refer to [14].

Though, also here, network disruptions and the moving of nodes in between subnets, plus the fact those subnets may never reunite, render this approach inappropriate for use in MIDAS.

Note too that in MIDAS we do not just need to know the *order* of events for conflict resolution, but we also need to be able to relate timestamps as closely as possible to the actual time of day. This is because such data can be of direct interest to users (e.g. hospital staff might want to know not only that casualty received morphine *after* he received a weak pain killer, but also *how long ago* – compared to the current time – the morphine was given). It might also be that it is necessary to generate an official event log for legal reasons.

Summing up: clock synchronisation is not a feasible solution in the environment envisaged in MIDAS because clock synchronisation takes time, there is no global reference, network partitions may occur, and a relationship to real-world time is important.

Fortunately, as we will show, synchronisation of clocks it is not the only way to provide a common notion of time.

Assumptions about clock characteristics

In developing a mechanism for achieving a common notion of time, we make the following assumptions:

- Each node has its own, independent functioning clock.
- All clocks may go at different *speeds*. Although speed differences can be expected to be small, there is no need to exclude the possibility that they may exist. And note too that even small differences in speed may accumulate over time.
  Clocks F1 and S1 in Figure 1 show a fast clock and a slow clock respectively.

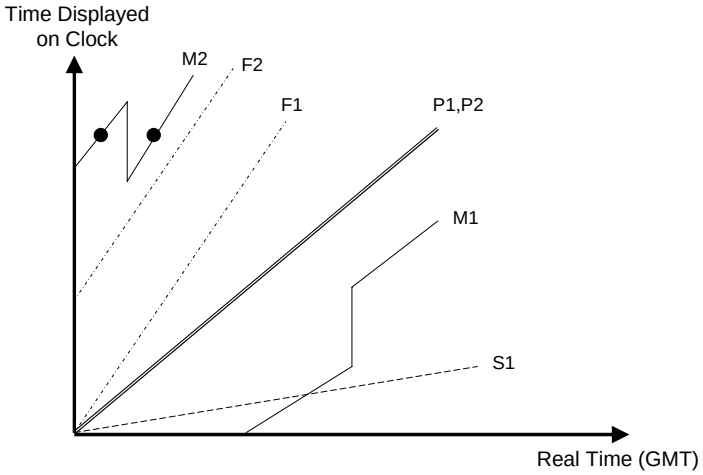

Figure 1: Clock characteristics

- Some clocks may be *ahead* or *behind* others (i.e. show a different time), even though may have the exact same speed.
  In Figure 1, clocks F1 and F2 are going at the same speed – but F2 is ahead of F1.
- It is not necessary to have any clocks that guarantee to show correct, absolute time (i.e. time based on GMT).
- A subset of the clocks may exist that can be defined as *perfect clocks*. We define a set of clocks as *perfect* if we can assume that they are synchronised with each other by some dependable external means (e.g. use of GPS time; use of the Network Time Protocol). A set of perfect clocks would typically be synchronised to real, absolute time (i.e. GMT), but it is not essential to our approach. The only relevance of perfect clocks is that they can be used to limit inaccuracies in the method of providing a common notion of time. The proposed approach works even if there are no perfect clocks.
  Clocks P1 and P2 in Figure 1 (lines drawn on top of each other) show two perfect clocks that are synchronised to GMT.
- Users may manually move the system clock forward on their device; this is really just a special case of a clock moving at a different speed from others. This has happened to clock M1 shown in Figure 1. Note that all times shown on the clock correspond to a unique real time.
- Users may *not* manually move the system clock backwards. The reason for this is that it could result in the same value being shown on the clock for two events which in fact occurred at different times in reality.
  This has happened in clock M2 shown in Figure 1. Note that there are times displayed on the clock (shown by dots in the diagram) that have the same value – but correspond to different real-world times.

The approach described in the next section aims to take account of all of the above cases.

Achieving a Common Notion of Time using Arrogant Clocks

Recall that, without a common notion of time:

- Applications cannot find out when a record was created
- Event logs (“global state”) can be generated – but the timestamps cannot be trusted, and the order of the events may not indicate the true order.

- It is not possible to base conflict resolution on the notion of the “latest” record.

Instead of trying to synchronise the clocks, or use logical or vector clocks, we propose to avoid the above limitations by following an alternative approach to achieving a common notion of time, as outlined in [1][2]. It is based on the idea that no attempt at all is made to synchronise clocks on different devices but that, instead, each device always considers that its own clock shows the “right” time. Because the approach is based on the idea that each clock thinks it is right and all the others are wrong, we call it “arrogant” clocks. This is somewhat akin to the way in which people in different countries in the world use different time zones – with each considering their time zone to be the “right” one for their purposes.

In order that such an approach can provide a *common* notion of time, each time a records is transmitted from one node to another during data synchronisation, the timestamp must be updated to the “time zone” used on the receiving node. This can be done quite easily by comparing system clocks at the moment at which the synchronisation takes place.

In order to implement this approach, some extra fields need to be included in the records transmitted between nodes during synchronisation, as follows:

my_time	Timestamp for the record, converted to local time on this node. This is the value that will be used for presenting information to the user, and for ordering of records for conflict resolution.
transmit_time	The time shown on the local clock of the node transmitting a record during a synchronisation operation – immediately before transmitting it.
sender_node	The unique node identifier of the node transmitting a record during a synchronisation operation.

We also need the following basic functions:

MY_CLOCK()	Returns the time currently shown on the system clock of the node calling the function.
MY_NODE_ID()	Returns the unique node identifier of the node calling the function.
TRANSMISSION_DELAY(*n*)	Returns the estimated time to transmit a message from the node with unique identifier *n* to the node calling the function.

Using these building blocks, we can describe the approach in simple pseudo code as follows:

```
When the record is originally created:
        my_time := MY_CLOCK()

Immediately before a synchronisation record is transmitted:
        transmit_time := MY_CLOCK()
        sender_node := MY_NODE_ID()

Immediately after a synchronisation record is received:
        clock_difference := MY_CLOCK()-transmit_time-
                            TRANSMISSION_DELAY(sender_node)
        my_time := my_time + clock_difference
```

Note that the approach depends crucially on the calculation of “clock_difference”: it is this value that is used to determine the difference (at the time at which the synchronisation took place) between the clocks on both nodes. To make this as accurate as possible, it is essential that the value of “transmit_time” is entered in the record at the very last possible moment, immediately prior to transmission. Similarly, invocation of MY_CLOCK() must occur on the receiver node as soon as possible after receipt of the record. There may be some delays at both sides, and these can be included in the TRANSMISSION_DELAY estimate.

Note that synchronisation in MIDAS is not just between pairs of nodes, but may involve propagation of records via multiple synchronisations across the network. Thus, errors in “clock_difference” may accumulate with such multiple synchronisations.

In order to minimise the effect of such accumulated errors, the approach can be refined to exploit the presence of any perfect clocks in the network. The aim is to make sure that, any time the record is propagated via a node with a perfect clock, the time calculated there is stored and used as “my_time” in any subsequent nodes with perfect clocks - thus avoiding errors accumulated during propagation between other nodes. To do this, we need to add another extra field to the record:

Perfect_time	Timestamp for the record, after conversion to “my_time” on a perfect node. If the record has never synchronised with a perfect node, the value of this field is set to EMPTY. The field is used to make sure that, when a record propagates through a sequence of nodes, the same time is recorded on all perfect nodes – which is appropriate, as they are all synchronised.

We also need another basic function:

PERFECT()	Returns TRUE if the node calling the function has a perfect clock.

Finally, we modify the pseudo code as follows:

<u>When the record is originally created:</u>

Append the following:

```
IF PERFECT() THEN perfect_time := my_time
           ELSE   perfect_time := EMPTY
```

<u>Immediately before a synchronisation record is transmitted:</u>

<no change needed>

<u>Immediately after a synchronisation record is received:</u>

Replace with the following:

```
IF NOT PERFECT() THEN
    <same code as before>
ELSE IF perfect_time = EMPTY THEN
    perfect_time := my_time
ELSE
     my_time := perfect_time
```

Assessment

The proposed approach to achieving a common notion of time has the following advantages:

- It is conceptually simple, and easy to implement.

- It is independent of any central server, and can be applied in a network of nodes where communication is highly intermittent.
- It allows users on different nodes to view and record time using the system clock present on their device – which is what they use to relate to time in their real world.
- It is based on a very flexible set of assumptions regarding clocks.
- Adjustments for differences in time between clocks are carried out at the moment of synchronisation. Thus, if a pair of clocks drift further and further apart, the adjustments will adapt to this.

The weakness of the approach is that errors may arise due to inaccuracies in estimates of transmission times between nodes. However, such errors are likely to be in the order of milliseconds (typically), and unlikely to be greater than tens of seconds. Given the assumptions earlier about time criticality of data in the types of applications for which MIDAS is intended, this is not considered to be a major problem. Furtheremore, such errors can be limited by carrying out occasional measurements of the actual transmission delay between nodes, in order to estimate as accurately as possible.

The extensions to exploit perfect nodes can also be highly beneficial, as they limit cumulative error during propagation. It is likely that a significant number of nodes in the types of application envisaged could be perfect nodes, as more and more mobile devices are including GPS as standard (which allows for accurate, synchronised GMT time).

For the types of applications for which it is intended, the arrogant clocks approach provides a practical and effective – though not perfect – way of achieving a common notion of time.

Example

This example illustrates the arrogant clock concept by showing how the extra fields and pseudo code behave as some records are synchronised between nodes.

Some information on how to read this figure:

- The columns in Figure 2 indicate that Nodes A and D have perfect clocks (e.g. they use a GPS device), that Node B is always 50 seconds *ahead* of real time) and that Node C is always 50 seconds *behind* real time.
- The tops of the record boxes indicate the time at which the record was created on this node. (The height of the box is not significant). For instance: the first record on Node A was created at t=0, then replicated on Node B during a synchronization process that started at t=30 and ended at t=33.
- All records have an ID value indicating their origin: all replicas of the same original record have the same ID.
- Arrows $\bullet\!\!\rightarrow$ indicate that synchronisation is taking place. The round edge indicates from which node and at what time synchronisation started. The arrowed edge indicates at which node and at what time the information was received.
- For reasons of brevity, fields *transmit_time* and *sender_node* are not included in the figure, nor is the calculated value *clock_difference*. These are explained in detail in Table 1.
- In this example, we assume that TRANSMISSION_DELAY() always returns 1. In all the synchronizations, the actual transmission time shown is 3 seconds. Note that this means that a 2-second inaccuracy is introduced at each synchronization (apart from between perfect clocks).

Table 1 explains how the values are calculated in the example in Figure 2.

t *(Real time)*	Action	Supporting pseudo code for calculating field values
0	Node A creates record (ID=1)	*my_time* := MY_CLOCK() → *0* Node A has a perfect clock, so: *perfect_time* := my_time → *0*
30	Node A and Node B enter the same network and commence synchronisation. As the only available record is stored on Node A (with ID=1), this is synchronised to node B.	At Node A, right before synchronising: *transmit_time* := MY_CLOCK() → *30* *sender_node* := MY_NODE_ID() → *A*
33		Node B, right after synchronising: *clock_difference* := MY_CLOCK()-*transmit_time*-TRANSMISSION_DELAY(A) → 83-30-1 = *52* (the first value is 83 as it took 3 seconds for the message to arrive, making the real time of reception 33, though the local clock is 50 ahead, so 33+50. The second value is 30 as indicated by the *transmit_time* field in the record; the third value is the assumed transmission delay of 1). *my_time* := *my_time* + *clock_difference* → *52* Note that this value of 52 for *my_time* corresponds to t=2 in real time – which is 2 seconds *after* the time the record was actually created. The inaccuracy has arisen because the estimated transmission delay was 1, but transmission actually took 3 seconds.
50	Similarly to above: Node A synchronises with Node C (ID=1).	At Node A, right before synchronising: *transmit_time* := MY_CLOCK() → *50* *sender_node* := MY_NODE_ID() → *A*
53		Node C, right after synchronising: *clock_difference* := MY_CLOCK()-*transmit_time*-TRANSMISSION_DELAY(A) → 3-50-1 = -48 *my_time* := *my_time* + *clock_difference* → *-48*
90	Similarly to above: Node C synchronises with Node D (ID=1).	At Node C, right before synchronising: *transmit_time* := MY_CLOCK() → *40* *sender_node* := MY_NODE_ID() → *C*
93		Node D, right after synchronising: This node has a perfect clock. So the formula used is simply: *my_time* := *perfect_time* → 0 Note that this means that the 2-second inaccuracy introduced in Node C at *t*=53 has now been removed, by using the *perfect_time* value stored in the record at *t*=0.
100	Node B creates a new record (ID=2)	*my_time* := MY_CLOCK() → *150* No value in *perfect_time* as Node B does not have a perfect clock.
150	Node A creates a new record (ID=3).	*my_time* := MY_CLOCK() → 150 Node A has a perfect clock, so: *perfect_time* := my_time → 150
200	Node A and Node C synchronise (ID=3)	At Node A, right before synchronising: *transmit_time* := MY_CLOCK() → *200* *sender_node* := MY_NODE_ID() → *A*

203	This will result in two records for the same primary key being present on Node C.	Node C, right after synchronising: *clock_difference* := MY_CLOCK()-transmit_time-TRANSMISSION_DELAY(A) → 153-200-1 = -48 *my_time* := *my_time* + *clock_difference* → *102*
250	Similarly to above: Node B and Node C synchronise (ID=2).	At Node B, right before synchronising: *transmit_time* := MY_CLOCK() → *300* *sender_node* := MY_NODE_ID() → *B*
253	There will then be *three* records for the same primary key present on Node C.	Node C, right after receiving the sync message: *clock_difference* := MY_CLOCK()-transmit_time-TRANSMISSION_DELAY(B) → 203-300-1= -98 my_time := my_time+clock_difference → 150 – 98 = 52 Queries will use *my_time* to select which of the three records is the latest. This is the one with ID 3, which was created at *my_time* 102. This selection is correct, as it is this record that was created last in real time.

Table 1: Description of Figure 2

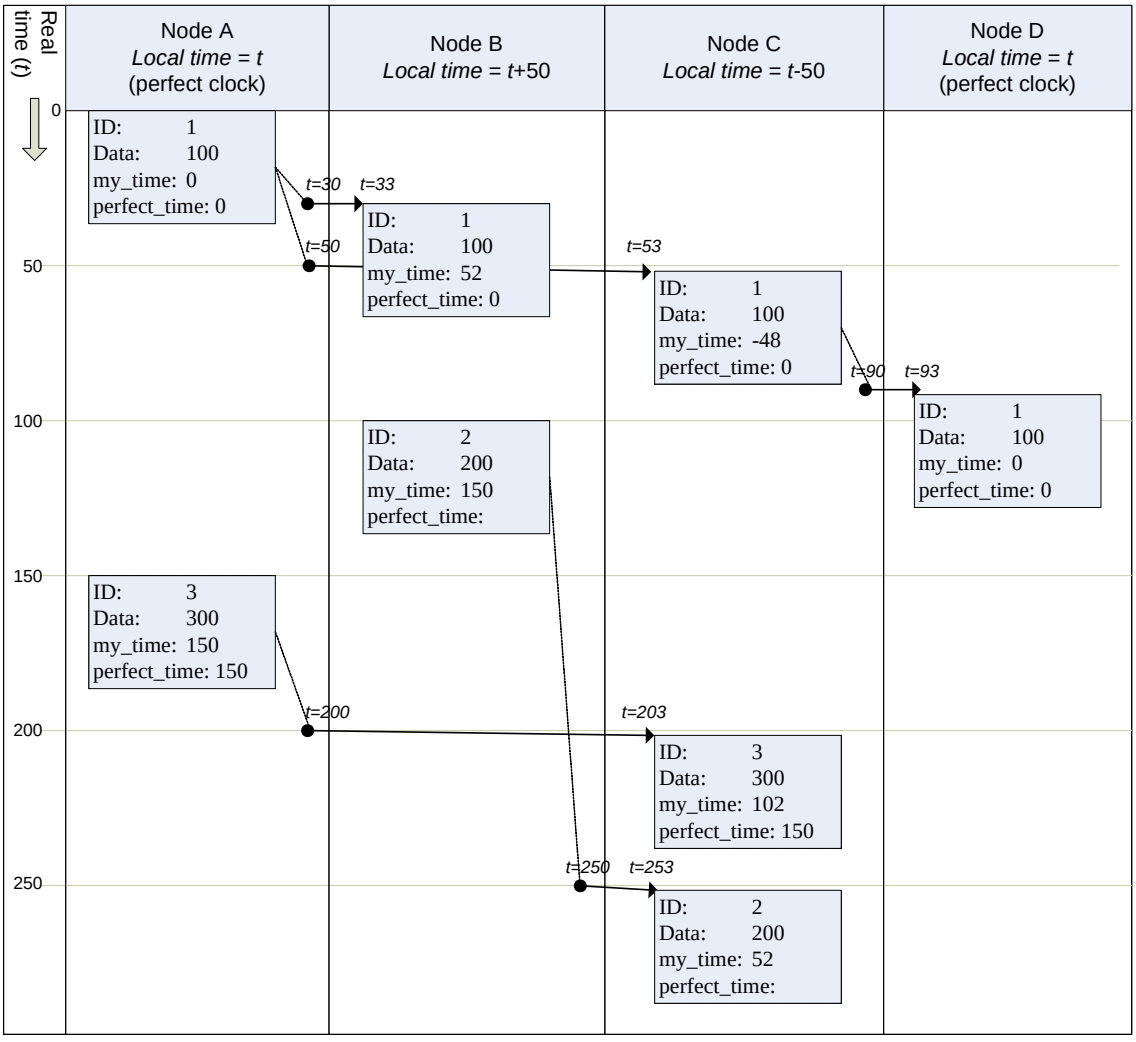

Figure 2: Synchronisation with arrogant clocks

Future Work

The natural next steps in this work are as follows:

- Devise a more formalised model of the approach, and use this to produce formulae which identify the factors influencing the accuracy of the time adjustments made during synchronisation.
- Build a simulator that simulates many nodes that have clocks set to different times, and have different fast/slow behaviours etc. These nodes must generate and replicate data and go through a series of network disruptions and reconnections between different sub-networks. Algorithms for generating, and an application for analysing event logs on the different node must be developed in order to assess the effectiveness of the approach.
- Produce a practical implementation of the approach, fully integrated in the MIDAS middleware, and used for automatic conflict resolution.

Acknowledgments

The MIDAS project receives financial support from the FP6 IST Programme of the European Commission, contract no. 27055.

References

[1] Joe Gorman, Håvard Kvålen, Ståle Walderhaug, "*Reliable Data Replication in a Wireless Medical Emergency Network*", SAFECOMP conference proceedings, Springer, Edinburgh 2003.

[2] Joe Gorman, Ingrid Svagård, *"Technology for Information Gathering and Sharing in Large-Scale Emergencies"*, Prehospital and Disaster Medicine, Vol. 19, Number 2, 2004.

[3] Alan Demers, Karin Petersen, Mike Spreitzer, Douglas Terry, Marvin Theimer, B. Welch, *"The Bayou Architecture: Support for Data Sharing among Mobile Users"*, Proceedings of the workshop on Mobile Computing Systems and Applications, Santa Cruz, California, December 1994

[4] Yasushi Saito, Marc Shapiro, *"Optimistic Replication"*, ACM Computing Surveys Vol. V No. 3 2005

[5] K.S. Skjelsvik, V.Goebel, T. Plagemann, *"A Highly Available Distributed Event Notification Service for Mobile Ad-hoc Networks"*, IEEE Distributed Systems Online Journal, Special Issue for Work-in-Progress Papers of Middleware 2004.

[6] N.Sanderson, V.Goebel, E.Munthe-Kaas, *"Knowledge Management in Mobile Ad-hoc Networks for Rescue Scenarios"*, Workshop on Semantic Web Technology for Mobile and Ubiquitous Applications, ISWC 2004, November,2004.

[7] T.Plagemann, J.Andersson, O.Drugan, V. Goebel, C.Griwodz, P.Halvorsen, E.Munthe-Kaas, M.Puzar, N.Sanderson, K.S.Skjelsvik, *"Middleware Services for Information Sharing in Mobile Ad-hoc Networks - Challenges and Approach"*, IFIP Workshop on Challenges of Mobility (WCC 2004), August 2004.

[8] T.Kristensen, S.Preben, L.Arnesen, E.Valen, T.Plagemann, *"Evaluation of middleware for distributed objects on handheld devices"*, International Workshop on Multimedia Interactive Protocols and Systems 2003, MIPS 2003, Napoli, Italy, November, 2003.

[9] D.Ecklund, V.Goebel, T.Plagemann, E.F.Ecklund, *"Dynamic End-to-End QoS Management Middleware for Distributed Multimedia Systems"*, ACM Multimedia Systems Journal, Fall 2002.

[10] David Mills, *"Network Time Protocol (NTP) -Specification, Implementation and Analysis"*, Internet Engineering Task Force, available at: http://www.ietf.org/rfc/rfc1305.txt

[11] J. Gorman, The MIDAS Project: Interworking and Data Sharing, Interworking 2006, Santiago, Chile, January 2007

[12] Leslie Lamport (1978). "Time, clocks, and the ordering of events in a distributed system". *Communications of the ACM* **21** (7): 558–565. doi:10.1145/359545.359563.

[13] Chandy, K. Mani; Leslie Lamport (February 1985). "Distributed Snapshots: Determining Global States of a Distributed System". *ACM Transactions on Computer Systems* **3** (1): 63–75. doi:10.1145/214451.214456. Retrieved on 2007-02-02.

[14] *Mattern, F. (Oct. 1988), "Virtual Time and Global States of Distributed Systems", in Cosnard, M., Proc. Workshop on Parallel and Distributed Algorithms, Chateau de Bonas, France: Elsevier, pp. 215-226*