DataWarp: Empowering Applications to Make Progress in the Face of Contradictory or Inconsistent Data

Stephen Crouch, Peter Henderson, Robert John Walters
University of Southampton,
Highfield, Southampton,
United Kingdom,
SO17 1BJ
{stc,ph,rjw1}@ecs.soton.ac.uk

Abstract

In modern systems data is distributed and replicated. Its ownership is also distributed amongst a variety of stakeholders with differing requirements and expectations. A consequence is applications inevitably find data they need is missing or contains contradictions and inconsistencies from time to time leaving them unable to progress within their domain. Techniques such as transactions can help but are not enough as they rely on being able to impose their own consistency rules on data and this isn't always possible. What is required is an ability for applications to move forward despite the problems in local data by applying higher-level reasoning to that data. This paper describes DataWarp, a technique which empowers applications to make progress in such environments and illustrates its use to specify an efficient personal Grid workflow scheduler.

1. Introduction

In the past when applications operated in isolation, each had complete control over the data they used and owned. For modern connected systems, this is no longer the case and now applications need to use and manage data which is distributed and replicated. Techniques such as transactions and systems of compensations approach the problem from the point of view that there is a single definitive value for every data item and they contrive to force this point of view onto data; their approach to solving problems of inconsistencies and other defects in data is to try to drive them out. They seek to achieve this by restraining and controlling the way that applications interact with data in such a way that global consistency is always maintained.

It would be nice if it were possible to eliminate all defects from data but, in reality, data in modern systems has many owners with differing outlooks and expectations. It is widely distributed and at least partially, replicated in many places. It is practically impossible to achieve permanent, global accuracy and consistency. The consequence is that applications inevitably encounter problems with the data they use from time to time and it is no longer acceptable for them to simply wait for such issues to be resolved.

In this paper we describe DataWarp [6, 10], an approach which permits applications to take control of their situation to make progress when faced with problematic data and illustrate applying the technique using to a personal workflow scheduler for the Grid.

2. Traditional Approaches

Traditional thinking about data which is replicated and distributed assumes that each datum has a "true" value and the data as a whole describes a world which, in general, is consistent and reasonable. Accepting this assumption, it follows that it should be possible for applications to avoid any difficulties arising from the data they use if they take sufficient care to ensure that all operations on data preserve its integrity (and compliance with the truth). This view of data

is embodied in the use of transactions and other approaches such as compensations.

2.1. Transactions

In a transactional system [8], whenever there is a change to data, all aspects of the change are made in a single logical step. The details of how this is achieved are beyond the scope of this paper, what is important is that from the point of view of an observer, the data moves from one consistent state to the next. If intermediate states do exist, perhaps where only part of the change has taken place, observers of the data are prevented from seeing them.

For a transactional system to achieve the goal of ensuring the whole body of data is always globally consistent, every transaction must include every copy of every datum which needs to be updated. If any is left out, the guarantee that consistency is maintained by the transaction is lost. This is restrictive and can lead to difficulties. For example, some copies of data might be located on portable devices which spend significant periods of time disconnected and the loss of a network link could easily result in some copies of data being unreachable. Either of these would cause transactions to fail and prevent applications from making progress. Even where it is possible to control data in this way, the necessity for transactions to have simultaneous and exclusive access to each and every data item they affect represents a considerable constraint on the actions of applications and obtaining access to data items which are used by many (or all) transactions can present a bottleneck.

2.2. Compensations

A less restrictive alternative to transactions is to adopt a system of operation which uses compensations [5]. Such systems are able to take actions which amount to partial completion of a transaction. They don't need the elaborate mechanisms of transactions to give data the appearance of moving instantly from one consistent state. Instead, applications proceed with their work step by step. Attached to each step is information about what to do if the application has to retract from an unacceptable situation. If the action is one which can be reversed, the compensation information might define how to "undo" the action. In other cases it would describe what action has to be taken to bring the application and data back into an acceptable state (not necessarily the same state as would have existed if the action hadn't been started). These "compensations" accumulate as the application proceeds. If something goes wrong, the application executes these compensations and hence restores itself and the data to an acceptable state. Alternatively, (eventually) some event will occur which will permit the application to identify that it has completed some set of actions at which point the compensations are no longer needed and can be discarded.

Compensations relax some of the demands made by transactions but in doing so, they expose intermediate states to observers and other users of data. If data is to remain globally consistent, it is still necessary for them to access every copy of every datum they use.

2.3. Shortcomings of traditional methods

Transactions, compensations and related schemes rely on the fundamental assumption that each datum has a "true" value, that every copy of it will have that value and that if all the data were collected together, the resultant collection would be free from contradictions and inconsistencies. Unfortunately, this is not quite true. In reality, data is not globally consistent for a number of reasons, including:

- Timing differences. Unless access to is strictly controlled using something like transactions, updates take time to propagate.
- Some data elements have values which change according to their context. For example, when asked their address an individual may respond differently if they at work or at home.
- Ownership, confidentiality and other issues may slow or prevent propagation of updated values.
- The extent and stringency of consistency requirements vary with the needs and outlook of the application. Data which is acceptable to one may be problematic for another.

3. An alternative approach: DataWarp

DataWarp seeks to relax the strong requirements associated with approaches to using distributed and replicated data such as transactions and compensations to permit applications to work in environments where the data they needed might not be available on time and could also be inaccurate or inconsistent [9]. The objective is to enable applications to work with data where there is no unified notion of the correct value for data items. It was inspired by the notion of virtual time in which the central system clock is replaced with a local notion of the current time in each application without affecting the results of the computation [11, 12].

When an application using transactions or compensations encounters data problems, it has to delay its work until the problem is resolved, possibly accompanied with an appeal for corrections to some appropriate authority.

However, today's large open systems have data which is subject to constant change, partially replicated in numerous locations and maintained for different purposes by numerous owners. There are many circumstances which can lead to data appearing inconsistent. Provided the data concerned isn't subject to constant update, problems arising from timing differences generally sort themselves out. Others may require external intervention but modern systems have no central authority able to arbitrate and resolve difficulties. The consequence is that an application which simply waits when it encounters problematic data can suffer considerable delay achieving its objectives and may never be able to complete any task.

In the DataWarp philosophy, the attitude is taken that the application has to take responsibility for getting its work done: it cannot afford to wait for some external actor to correct things and time spent waiting is time which could have been used doing something useful.

A heavy weight implementation of the DataWarp philosophy would be, every time a problematic data item is encountered, to take all data which are in doubt and calculate the consequences of adopting every possible value for each. The application can then construct the complete set of possible futures, creating a tree of possible computations in which each branch branches again each time a new data issue is encountered. It may be possible to prune out some branches as impossible. Of the remainder, one has to be the "correct" one – the one which would have happened if the data issue hadn't arisen. So long as the application doesn't have to reveal its state to the outside world, it can proceed in this indeterminate situation simultaneously progressing many candidate states. This set of states will grow each time data problems are encountered, and shrink whenever a problem is resolved.

Difficulties arise when the application has to do something which requires outside interaction. It is no longer possible to follow all possible executions when some dictate one action and others a different one. Now the application has to make a choice. From all possible actions, the application has to select one to implement. This could be the one which is considered to be most likely, but the selection can be made using any measure appropriate to the application. For example, the action which entails the least effort or is least likely to lead to danger or financial loss. Having selected an action, the application is able to continue but now when a data issue is resolved it should compare what it has actually done with what it can now see it *would have done if the data issue hadn't occurred*. Hopefully in most situations, the application will find that what it did is exactly "right" – or close enough not to matter (perhaps subject to some small adjustments). In the worst case, the application will have done something undesirable and it will have to retract or compensate for some of its actions to get back to a satisfactory situation – or as close as it can manage.

In practical application DataWarp applications don't calculate the full set of possible executions. Instead, they select a subset of likely candidates (possibly only one) based on an analysis of what the problems are in the data. The application needs to keep a record of actions which are provisional as they arise from data which is in doubt. However, it is to be hoped that in most instances each such action taken will turn out to belong to one the following three classes:

- It was exactly the right thing to do.
- It wasn't exactly right but it was close enough not to matter (not enough to warrant action, anyway).

- It was so long ago that it doesn't matter any more.

In the event that it turns out that the resolution of some data problem means the application is unhappy with what it has done, it has to do something about it. However, the hope and expectation is that when an application is operating in a familiar environment this will not happen often.

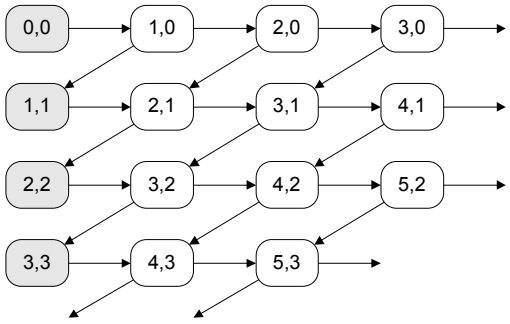

Figure 1: A single datum world

Figure 1 gives a graphical impression of DataWarp and other data management techniques in action. It describes a simple world in which there is just one datum (an integer). This datum is owned by an application which keeps it up to date by increasing its value from time to time. Also in this minimal world there are is another application which holds its own local copy of the datum. When its owner increments the datum it sends a notification to the other application but these notifications are not synchronised with the updates and sometimes take a considerable time to arrive. In the figure, the global state of the world is shown by the pair of values, the first being the "truth", the value held by the owner of the datum and the second being the value held by the other application in its copy. An update by the owning application alone causes the state to move to the right, and the arrival of the notification at the other application causes the system to move diagonally to the left and down as shown by the arrows. Clearly, for this world to be in a consistent state the two values must be the same and this would eventually occur if the data owner were prevented from making updates while the rest of the system continued to operate. If the system used transactions, consistency would be maintained and the world would only ever (be seen to) visit the states in the leftmost column. A compensation based system is able to move to the right from time to time, but is concerned to get back into the leftmost column and fires its compensations when it realises it has strayed to the right. In contrast, the DataWarp application recognises that data inconsistency is inevitable and normal, accepts this and is content to be in any column.

4. Using DataWarp to solve a Grid Scheduling problem

4.1. Workflows on the Grid

A typical Grid deployment scenario consists of an installed Grid platform, such as OMII [16] or Globus [7], acting as a gateway to a back-end large-scale computational resource [4]. An example of such a resource is a *batch processing system* (or resource manager) which manages a cluster of machines enabling jobs to be submitted from Grid clients and executed on those machines.

4.2. Typical Batch Processing Systems

The purpose of a batch processing system, or resource manager from a Grid perspective, is to schedule and initiate execution of batch jobs and to route these jobs between hosts. Many batch processing systems also have capabilities for transferring data between hosts.

Of the batch processing systems currently in use, many share common architectural features [1, 13]. A generic batch processing system may look like the one depicted in Figure 2 where there are the following structural concepts and entities:

- **Job:** submitted by the client, contains information on how to execute the job, and optionally includes additional details such as how/where to obtain the job's input and output and.
- **Job queue:** a list of jobs that are waiting to be executed.
- **Client(s):** user machine(s).
- **Execution node(s):** a host machine that accepts and executes jobs.
- **Server:** accepts and manages job requests.
- **Scheduler:** responsible for determining when and where waiting jobs are to be executed.

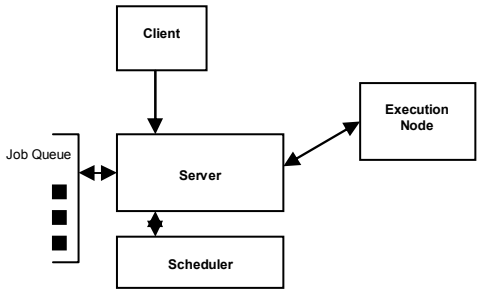

Figure 2: An overview of a generic batch processing system

Renderings of this framework differ across implementations. For example, a Condor [13] client has its own local scheduler that holds a local queue, and negotiates with a centralised server negotiator to schedule those jobs for execution whereas PBS [1] employs multiple queues held centrally on the server to hold waiting jobs with a centralised server scheduler. Two examples of PBS in real use are its deployments on the US TeraGrid and the UK National Grid Service (NGS) [15].

For this paper, we use the NGS as an example Grid deployment to illustrate a typical usage scenario. There are two key methods by which jobs may be submitted to PBS on the NGS: logging into one of the NGS head nodes and submitting to the PBS cluster accessible via that head node directly or using a Globus client installed on the user's machine to submit jobs through a Globus server. Either of these methods requires a valid set of credentials for authentication purposes to access the NGS through the Globus Security Infrastructure that ensures only approved users may use the resources. A full overview of these security measures is beyond the scope of this discussion. For the remainder of this paper we shall assume that users have the necessary credentials and are able to perform the two basic activities which enable them to use these resources; submitting jobs and monitoring the status of jobs.

Once jobs are submitted to PBS they reside on the queue until the scheduler allocates them to an execution node. There are two basic factors that affect how long this process will take to complete: Queue time (Q) and Execution time (E). Essentially, the time to execute the job will be Q + E.

However, it is possible to submit jobs which are not fully specified. For example, a job script may execute an application that uses data that is not yet available at the point of submission and it is possible to devise a process to take advantage of this situation to reduce the impact of Q. We can define this process with the concept of a placeholder job: a job which is inserted into the queue before it can be fully specified and proceeds up the queue as a normal job. At some point before it reaches the head of the queue, the job is re-configured. Users employ this tactic manually when the progress of jobs through the queue is slow but we are proposing to use the technique in an automated sense.

4.3. An Example Workflow

Typically, multiple job submissions are organised into a *workflow* that describes how the tasks are related in terms of control and/or data requirements and production.

For the purposes of this paper, let us introduce a simple example workflow described in pseudocode that represents a possible set of workflow actions. Such a representation allows us to abstract away from unnecessary implementation-specific details.

The workflow uses the following basic abstract structures, operations and concepts:

- *Data*: a structure that represents data.
- *Job*: a structure holding information about a job.

- *J.submitJob(Data D...)*: submits a job *J* to a batch scheduler with the set of data *D* as input.
- *J.waitFor()*: wait until the job has been executed
- *Data D = J.getResults()*: extract the results *D* from a job, *J*.

- *parallel* is used to denote concurrent execution of multiple branches.

The above functional abstractions are free to be defined in terms of a real platform; for PBS for example, *submitJob* would be based around the command *qsub*, and *waitFor* around *qstat*.

A simple example workflow with concurrency and conditional characteristics:

```
Data DI              # Input
Data DJK             # Output J or K
Data DA,DB,DC,DH     # Other Output
Job A,B,C,H,J,K      # Tasks
```

```
A.submitJob(DI)
A.waitFor()
DA = A.getResults()

parallel {
    B.submitJob(DA)
    B.waitFor()
    DB = B.getResults()
} and {
    H.submitJob(DA)
    DH = H.getResults()
    if ( some_predicate(DH) ) {
        J.submitJob(DH)
        J.waitFor()
        DJK = J.getResults()
    } else {
        K.submitJob(DH)
        K.waitFor()
        DJK = K.getResults()
    }
}

C.submitJob(DB, DJK)
C.waitFor()
DC = C.getResults()
```

This workflow which follows the typical Grid pattern of pre-process, compute and post-process is visually represented in Figure 3.

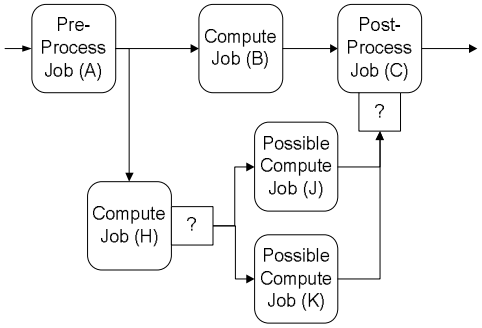

Figure 3: Diagrammatic representation of example workflow

4.4. Optimising a workflow using DataWarp

The simple obvious way to execute a workflow is to start at the beginning with the first task and put it in a queue for execution and wait for it to complete. Then, with that job completed, move on to the next… until you are done. The trouble with that is completion of a workflow with n elements entails n waits for a job to make its way up the queue [2, 14].

A nice feature about Grid workflows from the DataWarp point of view is that the typical workflow seeks to co-ordinate a number of activities to compile some result but these activities and their results are not often reported to or acted upon by others until the workflow is complete so our scheduler can exist in a non-determinate state with a number of possible executions in progress. It only has to finally choose which to keep at the last moment, just as it completes and announces the result.

The DataWarp scheduler is an application that an individual with a workflow to execute would use. Its objective is to complete a workflow as fast as it can; optimal use of resources and/or fairness of allocation is a problem for the scheduler at the data centre. The DataWarp scheduler uses DataWarp ideas at several levels.

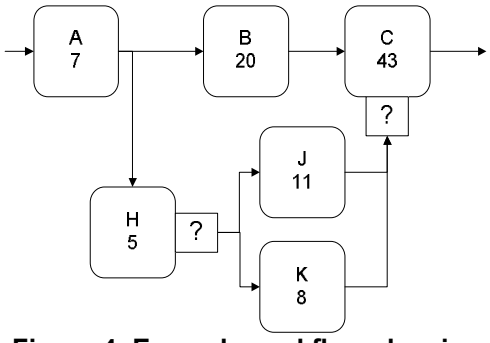

Figure 4: Example workflow showing execution times

Table 1: Job submissions timetable

Process	Execution Time (E)	Delay from start of execution for placeholder jobs
A	7	0
B	20	7
C	43	27
H	5	7
J	11	12
K	8	12

At the first level, the scheduler identifies the dependencies between the jobs a workflow contains. Any jobs which don't rely on results from others can be put on the queue for execution immediately (perhaps sorted into order of how early their results are likely to be used). Using experience of the behaviour of the queue and estimates of when required input will be available, the scheduler can add further jobs to the queue timed so that they arrive at the top just as the data they need becomes available. Getting this right permits all but the

first element of the workflow to proceed as fast as if it were possible to slip jobs straight into first place on the queue. Naturally, the scheduler needs to monitor its performance in this respect and be more or less aggressive in its placement of placeholder jobs as appropriate.

Taking the example from section 4.3, if it takes six minutes from submission from a job to begin execution and the jobs take the times shown in Figure 4, the scheduler would generate the job submission times shown in Table 1. Assuming execution proceeds according to plan, the workflow is completed by the DataWarp scheduler in 76 minutes (**6** + 7 + 20 + 43) compated with the naive execution which would take 88 minutes (**6** + 7 + **6** + 20 + **6** + 43).

At a second level, our scheduler can handle the consequences when things don't work out as planned. Consider the consequences if job *H* encounters difficulties and hasn't reported its result when *J* and *K* are scheduled to start. The placeholder jobs for *J* and *K* will arrive at the head of the queue before the scheduler is ready to decide which to execute. The DataWarp scheduler can handle this. It allows both to start, noting that it is still waiting for output from *H*. When it becomes available, the scheduler can abort the unnecessary job (or discard its output) and continue as normal.

Now let us consider the situation if *B* is finishing but *H* still hasn't completed. By now both *J* and *K* will be complete but the scheduler still can't decide which is the correct output to select as input to *B*. It can't do what it did with *J* and *K* because it only has one placeholder job ready. Instead, it can exploit knowledge of past executions of the workflow (or other metadata) to identify how critical this input is to the result. The scheduler may then be able to make a selection between the output of *J* or *K* and still complete *C* on time. In the example, the output from *J*, *K* could be a configuration for a visualisation programme with the property that, *J* always produces an acceptable result, whilst *K* produces one which is more aestheitcally pleasing but only works on some classes of data (identified in *H*). The scheduler can place a (backup) job in the queue for *C* which it can use if it becomes clear that its selection between *J* and *K* is unacceptable.

5. Conclusion

Modern applications operate in a connected world in which the data they use is distributed and replicated. A consequence is that it has become extremely hard, if not impossible, to ensure that data remains consistent and accurate. Schemes like transactions and compensations attempt to enforce consistency on data but are too restrictive to be used in general. The consequence is that applications need to be able to make progress in the face of defects in data since to do otherwise seriously compromises their ability to complete their work; they may wait indefinitely for data issues to be resolved and there is no central arbiter to whom to appeal for help.

DataWarp allows processes to make progress even when not all the data they require is unavailable or known to be inconsistent. It subsequently corrects any problems that arise, by making forward progress to an acceptable state, not by undoing its actions and we have outlined how this protocol can be applied to a personal Grid workflow scheduler.

In Grid applications, jobs typically take a long time to run and generally sit in a queue for long periods waiting to be executed. Using DataWarp, our scheduler is able to reduce the total time to complete a workflow by using placeholder jobs that will, ideally, reach the head of the queue just in time to begin execution. This approach is practical in the way the Grid is used today and is often implemented manually or in an ad-hoc workflow by users. How a community of users, all using such an optimistic scheduler once it had been automated, would evolve in practice is an interesting question. We believe the resulting resource utilisation would be acceptable from the resource owner's point of view and that the community of users would be seen to evolve into an ad-hoc collaboration, not unlike the competing participants in Axelrod's experiments [3].

6. References

[1] Altair Engineering Inc, "Portable Batch System." 2007, http://altair.org/.

[2] A. Andrieux, D. Berry, J. Garibaldi, S. Jarvis, J. MacLaren, D. Ouelhadj, and D. Snelling, "Open Issues in Grid Scheduling," in *Workshop on Open Issues in Grid*

Scheduling, National e-Science Centre, Edinburgh, 2003.

[3] R. Axelrod, *The Evolution of Co-operation*: Penguin, 1990.

[4] F. Berman, A. J. G. Hey, and G. C. Fox, *Grid Computing: Making the Global Infrastructure a Reality*: John Wiley and Sons Ltd, 2003.

[5] M. Butler and C. Ferreira, "A Process Compensation Language," in *Second International Conference on Integrated Formal Methods IFM2000*, Schloss Dagstuhl, Germany, 2000.

[6] S. Crouch, P. Henderson, and R. J. Walters, "Building Applications able to cope with Problematic Data using a DataWarp Approach," in *7th International Conference on Enterprise Information Systems*, Miami, USA, 2005, pp. 411-414.

[7] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scaleable Virtual Organization," *International Journal of Supercomputer Applications and High Performance Computing,* vol. 15, pp. 200-222, 2001.

[8] J. Gray and A. Reuter, *Transaction Processing: Concepts and Techniques* Morgan Kaufmann Publishers Inc,, 2001.

[9] P. Henderson, R. J. Walters, and S. Crouch, "Inconsistency Tolerance across Enterprise Solutions," in *8th IEEE Workshop in Future Trends of Distributed Computer Systems (FTDCS01)*, Bologna, Italy, 2001, pp. 164-169.

[10] P. Henderson, R. J. Walters, S. Crouch, and Q. Ni, "DataWarp: Building Applications which make Progress in and Inconsistent World," in *4th IFIP WG 6.1 International Conference, Distributed Applications and Interoperable Systems (DAIS 2003)*, Paris, 2003, pp. 167-178.

[11] D. R. Jefferson, "Virtual Time," *ACM Transactions on Programming Languages and Systems,* vol. 7, pp. 404-425, 3 July 1985 1985.

[12] D. R. Jefferson, "Virtual Time II: Storage Management in Distributed Simulation," in *9th Annual ACM Symposium on Principles of Distributed Computing*, Quebec City, Quebec, Canada, 1990, pp. 78-89.

[13] M. Litzkow and M. Livny, "Experience with the Condor Distributed Batch System," in *IEEE Workshop on Experimental Distributed Systems*, Huntsville, AL, 1990.

[14] M. Livny, J. Basney, R. Raman, and T. Tannenbaum, "Mechanisms for High Throughput Computing," *SPEEDUP Journal,* vol. 11, pp. 36-40, June 1997 1997.

[15] National Grid Service (NGS), http://www.grid-support.ac.uk/.

[16] Open Middleware Infrastructure Institute (OMII), "OMII Release," 2007.