Consistency based Snapshot management in Data Grids

Lutz Schlesinger
University Erlangen-Nuremberg
Department of Database Systems
Martensstr. 3, 91058 Erlangen
Germany
schlesinger@informatik.uni-erlangen.de

Wolfgang Lehner
Dresden University of Technology
Database Technology Group
Duererstr. 26, 01069 Dresden
Germany
lehner@inf.tu-dresden.de

Abstract

Almost over the last 20 years grid technology has been developed to exploit unutilized computing capacity around the world. The two major application areas are solving computing intensive problems having less data (computational grid) or operating on large volumes of data (data grid). From a database perspective it is advisable to replicate data sets at different nodes in the grid before executing a single query. The query processor assigns a query to those nodes, which do not operate at full capacity and which have stored the appropriate replica. As the replica may be outdated usually a synchronization mechanism is necessary. This may be extremely expensive if full consistency between original data and replica is required. To avoid synchronization we follow the approach to store multiple versions of static snapshots. New versions of local data sets are distributed to the different nodes in the grid and added to the set of locally stored snapshots. Holding multiple versions of snapshots, the user has the possibility to combine snapshots taken at different points in time to get a more globally consistent, but possibly older view. The access to outdated snapshots is acceptable, if the user has knowledge about the degree of staleness or the degree of inconsistency if unsynchronized replicas are combined for a global view. This paper focuses on the quantification of the inconsistency and the snapshot management at inidividual members of a global data grid.

1. Introduction

On the one hand statistics show that in the average only 5% of the CPU time of a desktop computer is used ([Bers02]). On the other hand many applications (e.g. *SETI@Home Project* SETI: Search for Extraterrestical Intelligence, http://setiathome.ssl.berkeley.edu/) need tremendous computing time to solve their specific problems. The internet consists of millions of computers and provides an infrastructure for pure data exchange. The idea of a *grid* ([FoKe99a]) as the new technology for the advanced Web ([Gent01]) is to build a network between computers with unutilized machine time to solve computing (*computational grid*, [FoKe99b]) or data (*data grid*, [MBM+99]) intensive problems. The *Global Grid Forum* (http://www.gridforum.org/) coordinates the standardization efforts (e.g. *Open Grid Service Architecture* (OGSA, http://www.globus.org/ogsa/)) and develops toolkits (e.g. *The Globus Toolkit* ([FoKe99c], http://www.globus.org/toolkit/) for an easier development of grid applications.

In the context of a data grid, applications operate on huge quantities of data, which are located on several usually geographically distributed nodes ([FoKe99a]). To be able to use unutilized computing power at run time, data is replicated before executing a single query ([Bers02]). The necessary infrastructural prerequisite is a high data transfer rate (> 10GB/s, [FoKe99a]), which becomes available nowadays (gigabit network over fibre). Data replication reduces data access latency by avoiding data shipping at run time and increases the performance and robustness. However, two major problems arise: One problem is locating and managing of replicated data sets. The other topic concerns maintaining the consistency between updated original data and replicas.

Related Work

Related work in the area of replica management proposes mostly high level concepts for distributing and locating replicas. For example [GLK+02] describes general requirements for a replica management service in the grid. [BCC+02] presents an overview of necessary functions in general and introduces a *gridOpen()*-command for locating replicas in more detail. A framework for a replication service and a prototypical implementation, named *Giggle*, is presented in [FIR+02]. [GLK+02] introduces the idea of a master copy in combination with a two-stage update process implemented in the *Reptor* project. In this approach the master copy is modified and the replicas remain in an old state leading to an inconsistency between master copy and replicas. In a second step the replicas are asynchronously updated by using the changed master copy. Finally Cameron ([Came02], the *Optor* project) locates the replicas on those nodes where they are most probably used. Finding a replica with lowest transfer costs is based on an auction. Common to all is an enhanced version of FTP ([ABB+02]) as protocol for efficient data transport, e.g. *GridFTP*, as well as replica catalogues (e.g. [StHa02]).

Contribution

In this paper we focus on the problem of avoiding the extremely expensive synchronization process by extending the idea of [GLK+02]. The main idea is to locally store the replica as snapshots, which are not synchronized with the master copy. To establish global consistency from time to time a snapshot of the master copy is locally stored once in a while, which avoids the complex process of asynchronously updates. If the new snapshot is added to the set of existing snapshots instead of replacing an old one, a version history of snapshots is set up. This enables the user to build an almost global, but possibly older consistent view by selecting different snaphots in time, if an access to outdated replicas is acceptable. To quantify and to compare different possible selection methods of snapshots a specification of the inconsistency is introduced in section 2. As the local storage for the snapshots is limited a management mechanism with regard to the inconsistency is introduced in section 3. Section 4 closes with a short summary.

2. Specification of Data Set Inconsistency

To provide the service of distributed query execution with lowest cost and to pick snapshots with different age algorithms for a parameterized selection of snapshots and operations between the selected snapshots are needed. At each node the existing snaphots can be illustrated in an object-time-diagram as shown in figure 1, where the time represents the valid time of the snapshots.

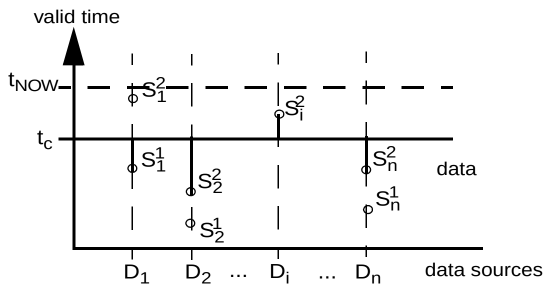


Fig. 1: Example for an object-time-diagram at a single node

Historic Cut

In a classical database middleware approach the snapshots reflecting the current state are selected and joined together. The selected snapshots are the closest snapshots to the horizontal line t_{NOW} reflecting the current time in the object-time-diagram. In our approach, where an arbitrary number of snapshots of the same data source exists, a selection at any time t_c ($t_c \leq t_{NOW}$), called the *historic cut*, is possible. In the optimal case all snapshots are valid at the same time (figure 2a) while in our data grid scenario the snapshots may have different valid times. Therefore, the connecting line t_c is no longer a straight line (figure 2b). The curve reflects the inconsistency relating to the different valid times. A metric to quantify the inconsistency is defined in the following. Algorithms for selecting the snapshots under consideration of the metric and the age of the snapshots are discussed in [ScLe02].

Quantifying the Inconsistency

Basically the inconsistency metric considers the time and the data change rate of a set of snapshots. Disregarding the second aspect for a moment, the time inconsistency for a single data source is defined as the distance between the

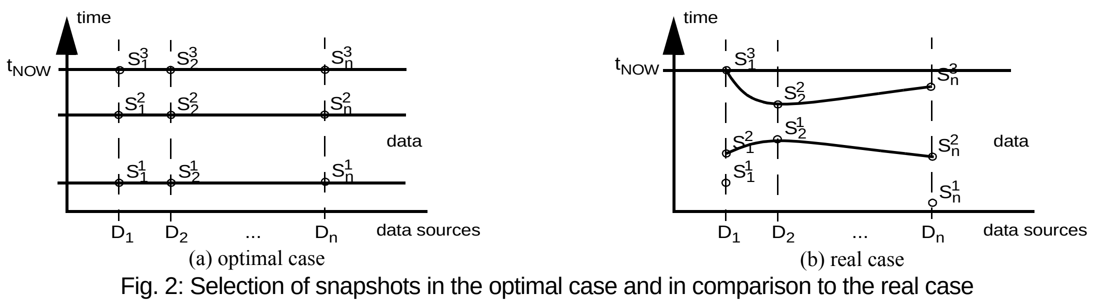


Fig. 2: Selection of snapshots in the optimal case and in comparison to the real case

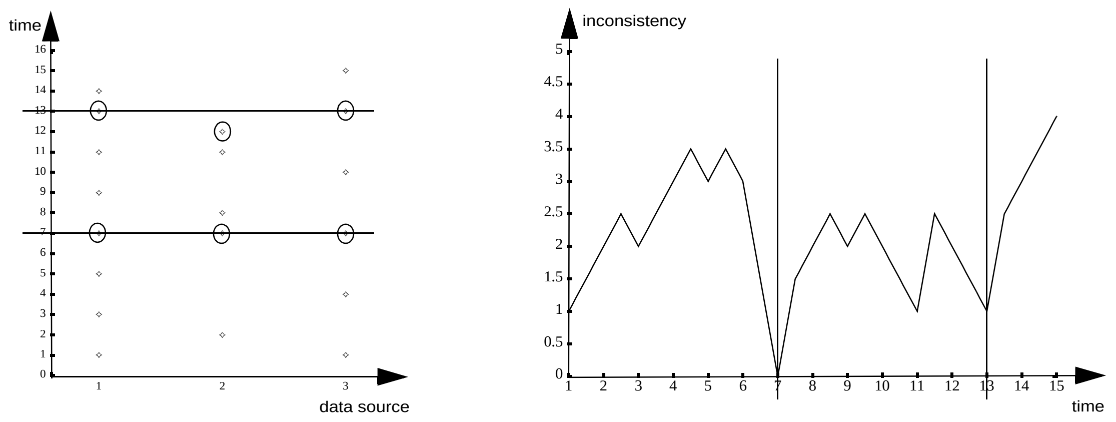


Fig. 3: Example for the selection of historic cuts and the resulting inconsistency curve

valid time of the snapshot and the time point of the historic cut. For *k* selected snapshots the inconsistency *I* is defined on the basis of the L_p-metric:

$$I = \sqrt[p]{\sum_{i=1}^{k} (time(S_i^j) - t_c)^p}$$

The example of figure 3 shows snapshots of three data sources in an object-time-diagram (figure 3 left) with $p = 1$. According to the formula the inconsistency at point 13 is 1 and 0 at point 7 (figure 3 right). This simple example illustrates the conflict between age and inconsistency: At point 7 the inconsistency is lower than the inconsistency at time point 13, but the age is much higher.

While the above inconsistency formula is based on the distance in time, the formula may be extended to additionally consider the data change rate. For each data source *i* we introduce a data change rate Δd_i reflecting the data changes of the object. This signature considers existential changes (insertions and deletions of objects) and has values between 0 and 1 (0-100%). This data change rate is a parameter of the reusage rate ρ at a time point *t* with t_s as the time point of the globally oldest snapshot:

$$0 < \Delta d_i \le 0,5 : \rho(t) = \frac{-1}{|t - (t_{NOW} - t_S) - 1|^{\frac{1}{2\Delta d_i} - 1}} \bullet \left(\frac{1}{t_{NOW} - t_S} \bullet (t - (t_{NOW} - t_S)) + 1\right) + 1$$

$$0,5 \le \Delta d_i < 1 : \rho(t) = \frac{1}{(t+1)^{\frac{-1}{2(\Delta d_i - 1)} - 1}} \bullet \left(\frac{-1}{t_{NOW} - t_S} \bullet t + 1\right)$$

At time point t_{NOW}, ρ has a value of 100% independendly of Δd_i and at time point t_s the value is 0. The value pattern between these points is determined by Δd_i and illustrated in figure 4. The combination of the distance in time and the reusage degree leads to an extended inconsistency formula, where α_k denotes weights for each single data source and $S_k^{j_k}$ are the selected snapshots:

$$I(time(S_1^{j_1}), \ldots, time(S_k^{j_k})) =$$

$$= \sum_{k=1}^{n} (\alpha_k \bullet |(time(S_k^{j_k}) - t_c)| \bullet (1 - \rho_k(time(S_k^{j_k}))))$$

The following two cases are very interesting:

- *$\rho = 1$, $\Delta t = 0$ and $\rho \neq 0$, $\Delta t = 0$ (for all snapshots)*
  Since the time of each snapshot is equal to the time of the historic cut, the selected snapshots are always most up-to-date and the inconsistency is equal to 0 independendly of the reusage degree.
- *$\rho = 1$, $\Delta t \neq 0$ (for all snapshots)*
  The snapshots are older than the historic cut. Since the reusage degree is 100% (no data changes), an access to old snaphshots corresponds to an access to snapshots, which have the time of the historic cut. Therefore, the inconsistency value is 0.

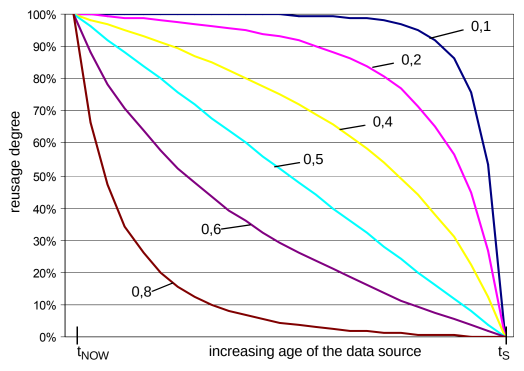
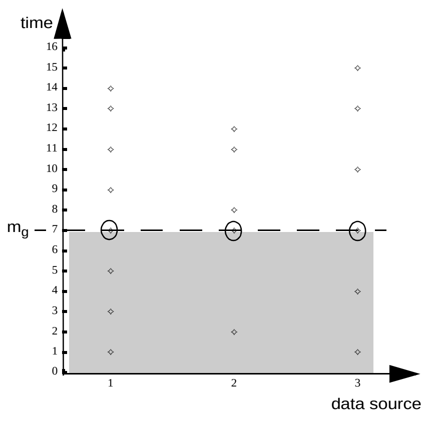


Fig. 4: Dependency of reusage degree and age



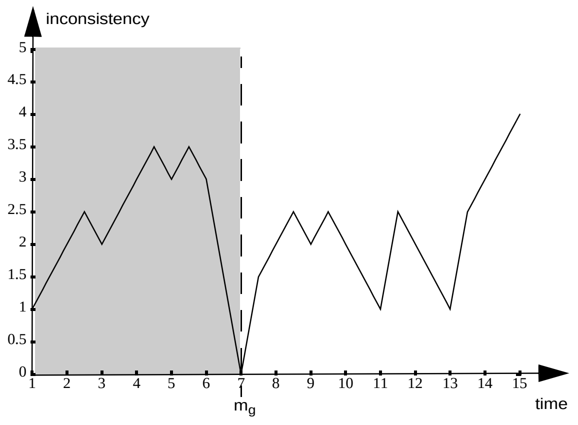


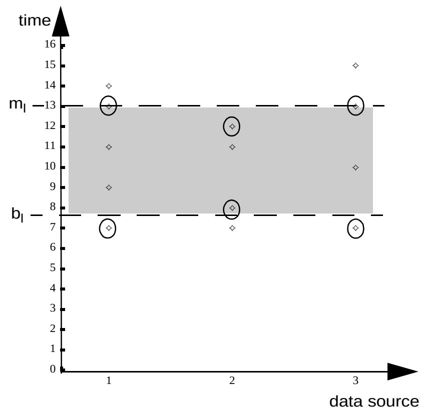
Fig. 5: Example of removing snapshots during the first step

3. Local Snapshot Management

As the local buffer for the snapshots is limited to a maximum size B_{max} a replacement strategy considering the inconsistency is presented in this section. On the basis of the inconsistency graph a probably user behaviour may be recognized. This influences strongly the proposed strategy. It removes snapshots in an iterative manner until the currently used buffer B_{cur} plus the size of the new snapshot B_{new} is smaller or equal to B_{max} (buffer constraint: $B_{cur} + B_{new} \leq B_{max}$). After each step the buffer constraint is checked and the algorithm stops if the condition holds.

In the first step the existence of a lower bound for selecting the historic cut is used because a user would never select a historic cut being older than the lower bound. This bound is determined by the global minimum m_g of the inconsistency graph. If more than a single time point exists with the same inconsistency value, then the point with the higher timestamp is selected. The reason for m_g as the lower bound is caused in the fact that a user would never choose a historic cut which is older and more inconsistent than a more up-to-date and more consistent point m_g. Therefore, all snapshots being older than m_g and not contributing to m_g may be freely removed. Continuing the example of figure 3 in figure 5 right, the time point 7 is the global minimum and all snapshot in the shaded area of figure 5 left are eliminated except of the snaphots marked by a circle.

The second step starts with finding the local minimum m_l most current. The inconsistency is denoted with $I(m_l)$ at m_l. All snapshots with the following property are no longer of any interest for the user: Either the snapshots are older than m_l or they contribute to an inconsistency value higher than $I(m_l)$. This is caused by the assumed user behaviour in two directions: A user selects a newer historic cut than m_l and accepts a higher inconsistency value. Alternatively a much older historic cut is only selected if the inconsistency value is smaller than $I(m_l)$. The area involved being of no interest for the user is determined by the points m_l and b_l. The point b_l is older than m_l, but closest to m_l and the inconsistency $I(b_l)$ is equal to $I(m_l)$. This point can be determined only algorithmically, because the function for computing the inconsistency is not reversible. In figure 6 right the grey shaded area marks the range of the inconsistency curve between b_l and m_l. The snapshots in the grey shaded area of figure 6 left are the corresponding candidates. From this set those snaphots are eliminated which are selected snapshots at historic cuts m_l and b_l. These snapshots are marked by a circle. The remaining snapshots can be removed.



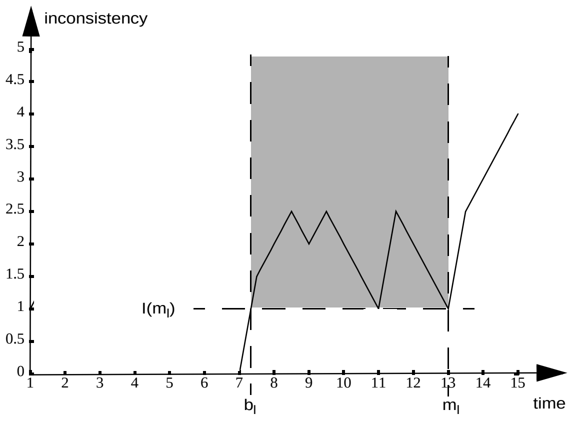


Fig. 6: Example of removing snapshots during the second step

If B_{cur} is still too small to store the new snapshot, snapshots are stepwise eliminated in the following manner: As for each data source at least the newest snapshot should be kept, the remaining snapshots are removed in ascending order of their benefit, which is the quotient of age and size of the snapshot. The final step depends on the data source of the new snapshot: If the new snapshot is from a data source where no old snapshot is available then the snapshot is stored. Otherwise an existing snapshot of the same data source is replaced by the new snapshot. In both cases, the buffer must have a capacity to store this snapshot, otherwise the buffer is too small and the storage size is automatically extended or the system administrator is informed.

In comparison to traditional buffer replacement algorithms ([GrRe93], [RaGe02]) this incremental algorithm removes possibly more than one snapshot in the first and second step and extends automatically the buffer in the third step. The selection of snapshots is similiar to the problem of generating the optimal combination of aggregates resulting in lowest costs in the context of materialized view selection ([HaRu96], [BaPT97]).

4. Summary and Future Work

In data grids, data sets are replicated and stored at different nodes to use unutilized computing power during query processing. As the data from different data sources are distributed before query processing it might be possible that they reflect different points in time. If old replicas are replaced by new replicas and stored as snapshots the user can select different snapshots at the same node. The quantifying of the appearing inconsistency by selecting older snapshots and the buffer management of the snapshots is discussed in this paper. The proposed concept and algorithm are currently under implementation within the *SCINTRA* (*semi-consistent integrated time replicated approach*) project. Future work in the area of snapshot management deals with the extension of the replacement algorithm by considering semantic connections as well as referential integrities between the snapshots of different data sources and the availability of data sources as well as the restoring of old snapshots.

References

ABB+02 Allcock, B.; Bester, J.; Bresnahan, J.; Chervenak, A. L.; Foster, I.; Kesselman, C.; Meder, S.; Nefedova, V.; Quesnel, D.; Tuecke, S.: Data Management and Transfer in High-Performance Computational Grid Environments. In: *Parallel Computing 28(5)2002*, pp.749-771

BaPT97 Baralis, E.; Paraboschi, S.; Teniente, E.: Materialized Views Selection in a Multidimensional Database. In: *23rd International Conference on Very Large Data Bases (VLDB'97, Athen, Greece, Aug. 25.-29.)*, 1997, pp. 156-165

BCC+02 Bell, W. H. ; Cameron, D. G. ; Capozza, L.; Millar, P.; Stockinger, K; Zini, F.: Design of a Replica Optimisation Framework. *Technical report*, DataGrid-02-TED-021215, Geneva, Switzerland, December 2002

Bers02 Berstis, V.: Fundamentals of Grid computing. *IBM Redbooks Paper*, IBM, 2002 (electronic version: http://www.redbooks.ibm.com/redpapers/pdfs/redp3613.pdf)

Came02 Cameron, D.: Replica Management and Optimisation for Data Grids. *Graduate Report*, University of Glasgow, UK, September 2002

FIR+02 Foster, I.; Iamnitchi, A.; Ripeanu, M.; Chevernack, A.; Deelman, E.; Kesselman, C.; Hoschek, W.; Stockinger, H.; Stockinger, K.; Tierney, B.: Giggle: A Framework for Constructing Scalable Replica Location Services. *Case Study*, University of Manitoba, Winnipeg, Canada, 2002

FoKe99a Foster, I.; Kesselman, C. (eds.): *The Grid: Blueprint for a New Computing Infrastructure*. Morgan-Kaufmann, 1999

FoKe99b Foster, I.; Kesselman, C.: Computational Grids. In: Foster, I.; Kesselman, C. (eds.): *The Grid: Blueprint for a New Computing Infrastructure*. Morgan-Kaufmann, 1999, pp. 15-51

FoKe99c Foster, I.; Kesselman, C.: The Globus Toolkit. In: Foster, I.; Kesselman, C. (eds.): *The Grid: Blueprint for a New Computing Infrastructure*. Morgan-Kaufmann, 1999, pp. 259-278

Gent01 Gentzsch, W.: Grid Computing: A New Technology for the Advanced Web. In: *Advanced Environments, Tools, and Applications for Cluster Computing, NATO Advanced Research Workshop (IWCC01, Mangalia, Romania, September 1-6)*, 2001, pp. 1-15

GLK+02 Guy, L.; Laure, E.; Kunszt, P.; Stockinger, H.; Stockinger, K.: Replica management in data grids. Technical report, *Global Grid Forum Informational Document*, GGF5, Edinburgh, Scotland, July 2002

GrRe93 Gray, J.; Reuter, A.: *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, 1993

HaRu96 Harinarayan, V.; Rajaraman, A.; Ullman, J.D.: Implementing Data Cubes Efficiently. In: *Proceedings of the 25th International Conference on Management of Data (SIGMOD'96, Montreal, Quebec, Canada, June 4.-6.)*, 1996, pp. 205-216

MBM+99 Moore, R.W.; Baru, Ch.; Marciano, R.; Rajasekar, A.; Wan, M.: Data-Intensive-Computing. In: Foster, T.; Kesselman, C. (eds.): *The Grid: Blueprint for a New Computing Infrastructure*. Morgan-Kaufmann, 1999, pp. 105-129

OGSA Open Grid Service Architecture web page, http://www.globus.org/ogsa/

RaGe02 Ramakrishnan, R.; Gehrke, J.: *Database Management Systems*. McGraw-Hill, 2002

ScLe02 Schlesinger, L.; Lehner, W.: Extending Data Warehouses by Semi-Consistent Database Views. In: *Proceedings of the 4th International Workshop on Design and Management of Data Warehouses (DMDW'02, Toronto, Canada, May 27)*, 2002, pp. 43-51

StHa02 Stockinger, H.; Hanushevsky, A.: HTTP Redirection for Replica Catalogue Lookups in Data Grids. In: *ACM Symposium on Applied Computing (SAC2002, Madrid, Spain, March 10-14)*, 2002