Versioning and Consistency in Replica Systems

Hartmut Kaiser[1], Kathrin Kirsch[2], Andre Merzky[3]

[1] Center for Computation & Technology, Louisiana State University
[2] Max Planck Institute for Psycholinguistics, Nijmegen
[3] Vrije Universiteit, Amsterdam

Abstract. Grid Replica systems are gaining foothold in real end user systems, and are used in an increasing number of large scale projects. As such, many of the properties of these systems are well understood.
This paper how to handle some minor shortcomings of todays data replica systems, in respect to consistency management, and to their ability to handle derived data sets. We think that both features will allow replica systems to gain wider acceptance in the GIS community.

1 Introduction

Distributed replica systems add real value to large data centric projects: an inspection of success stories like GriPhyN [1], LIGO [2,3] and the CERN Data Grid [4] show the central role replica systems play in their overall architecture. It is interesting that the number of *basic* concepts found in these replica system is small compared to the overall number of concepts provided by modern systems such as SRB [5,6]. Here, basic concepts are those provided by all replica systems and minimally *required* by all data management use cases [7,8].

- hierarchical logical name space with attributes (meta data)
- attribute access and manipulation
- data access and manipulation
- distributed architecture
- latency management
- back end system support

That list is, for example, completely implemented by the Globus Replica Location Service RLS [9,10], the CERN Replica Management System Reptor [11,12], the Storage Resource Broker SRB [5,6], and others. Successful deployment of these systems dominate todays landscape of data grids [1,2,3,4,13,14].

There exists, however, a set of use cases which, with the above set of properties, fail to be implementable, at least for large scale projects where scalability and maintainability are of increasing concern. We present 2 of these use cases in the next section, review the properties of existing replica systems in respect to these use cases in Sect. 3, and describe required additional properties in Sect. 4. We will propose a simple implementation on top of existing replica system in Sect. 5.

2 Discussion of Replica System Use Cases

2.1 Linguistic Scenario

The linguistic field of language acquisition is, although not commonly known, very data intensive and depends heavily on language corpora: for the linguistic analysis, human speech has to be recorded, transcribed and prepared as input for analysis tools. These steps are supported by Language Resource Archives (LRA), which also play a crucial role in development and testing of linguistic models, but also in language documentation and preservation [15,16].

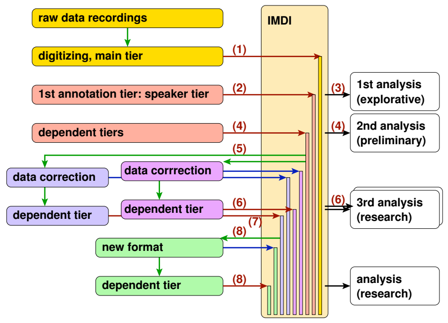

Fig. 1. The life cycle of linguistic data collection (simplified) can span years and is a large scale collaborative effort.

The resource intensity of data collection and preparation, especially in longitudinal language acquisition studies, encourages *data sharing* in the linguistic community. This however results in branching of the original corpus and a redundancy of corpus work, as the technical infrastructure of existing LRA supports only one main version (usually the one of the corpus owner), in a specific format and layout (determined by the used LRA). The following use case (also shown in fig. 1) describes requirements for *data sharing* in a language acquisition scenario, but translates easily to other fields such as language documentation and preservation, which have similar needs for collaborative data management[4].

(1) *The recording of a longitudinal language corpus takes several months or years, depending on the design of the linguistic study. Video and audio tapes are digitized, in accordance with the format requirements of the used LRA, and form the first version of the main tier of the new corpus in that LRA.*

(2) *The first annotation, a transcription of the audio tier, is added, again conforming to the LRA standards.*

(3) *A first preliminary and exploratory data analysis is performed, allowing for planning of promising research topics, and motivating the addition of further, research-specific annotation tiers.*

(4) *New dependent tiers are added to the corpus (usually morpho-syntactic tiers), and allow a first detailed research analysis.*

(5) *Different research groups obtain limited access to the data. Data are not prepared for publication at this stage, so data sharing occurs in controlled conditions: read and write access control is required for all tiers.*

(6) *New tiers are added which allow for a final analysis of the data, in respect to the various research questions motivated by the preliminary analyzes in (3), (4) and (5). This causes a split of the corpus, as changes and additions on the main tier or on dependent tiers are caused by different research objectives.*

(7) *A public version of the corpus is prepared by several groups which obtained access in (5). That public version allows a variety of research groups to use the corpus and associated tiers for their work.*

[4] All steps described in this use case are very time intensive and expensive, as there are no automatic or semi-automatic ways to transcribe spoken child language.

(8) Various tiers get subsequently added to the corpus by the public, and existing tiers continue to be changed by various groups, in various formats, increasing thereby the diversity, quality and usability of the published corpus.

At the present, LRA's only publish corpora as described in (7), as a single, cleaned, final and static version. The requirements vary with the LRA, but require significant effort, so that (a) only a small percentage of the data is made publicly available and (b) corpora are usually published years after their collection. The branches described in (5), (6) and (8) are not available for research as the LRA's do not allow for the coexistence of different corpus versions. Considering the intended corpus life time and usage (the linguistic community still uses data that were recorded in the 70s) this is an unfortunate situation.

The extension of LRA's with collaborative features will lower the entry requirements for smaller language collection projects, and would help to open their widely dispersed data collections to the community. It also would allow them to use the collaborative features of LRA's very early in the corpus lifetime.

2.2 Data Management in Geographic Information Systems

GIS draw their input data from a large variety of resources [17,18], which also implies a large variety of ownerships and copyrights [19]. Further, the transformation of these data, as required by any typical GIS system [20], imply the creation and maintenance of derived data sets, which, in our opinion, leads to similar collaborative requirements as the described linguistic use case.

The use case presented here is based on a scenario from the SCOOP hurricane forecasting project [21] (SCOOP: SURA Coastal Ocean Observing and Prediction). The Scoop-GIS has the goal to provide automated regular forecasts for gulf area hurricane tracks and storm surges. It models a data driven workflow using a central data archive as data repository for incoming, intermediate and resulting data sets. Although it is an realistic use case, it is not planned to implement it anytime soon: the SCOOP community is too small to be concerned about scalability. In particular, contrary to the scenario below SCOOP does not perform data replication and does not use collections.

(1) The National Hurricane Center in Miami, Florida (NHC), part of the National Oceanic and Atmospheric Administration (NOAA) provides the main input data for any hurricane forecast process. If a hurricane forms, the NHC provides track information for its 'center of pressure' in a 6 hour rhythm.

(2) NHC starts the execution of the workflow by placing the track data into the archive. That triggers the University of Florida (UF) which derives 4 additional tracks from the original NHC track by rotating that about ± 5 *and* ± 10 *degree. The UF also calculates wind fields for all 5 tracks, and adds the wind fields to the archive. Meta data annotations are used to maintain dependency information between the various data sets.*

(3) The placement of the wind fields triggers large simulation codes for various aspects of the cyclone system, for each wind field: Wave Watch 3 (WW3)

simulations predict ocean wave amplitudes; ADvanced CIRCulation models (ADCIRC) predict storm surges in coastal regions, etc.

(4) *The long running codes from (3) predict up to 72 hours of the cyclones development. Simulations snapshots are taken after 6 hours and are used as more realistic boundary conditions for the next run of the workflow. The simulation models hence need a couple of iterations to stabilize.*

(5) *Additional wind fields are sometimes provided by 3rd parties, e.g. by the Naval Research Laboratory of the US Navy (NRLMRY) or by the National Center for Environmental Prediction (NCEP). These wind fields start again predictive WW3 and ADCIRC simulations, but are, due to their singular inputs, not subject to the iterative boundary adjustment described in (4).*

(6) *The prediction outputs are stored in the data archive, and are used by SCOOP members and 3rd parties for automated or interactive visualizations, combining both static data sources (e.g. topological data), the original track data, and the predicted cyclone behavior over the next 72 hours.*

(7) *Both the 72 hour forecasts and the intermediate forecast results (such as the 6-hour snapshots) can be used to spawn off regional forecasts, leading to downscaled versions of step (3), (4) and (6).*

2.3 Basic Operations in the Use Cases

The basic operations required in the presented GIS scenario are similar to those required by the linguistic scenario: derived data sets play a central role, and versioning, consistency, and access policies need to be addressed, by maintaining the scalability of the overall system.

Versioning and Consistency: A change to a data set creates a derived data set. Similar as in Concurrent Version Systems (as CVS or Subversion), such changes may get merged into the original data set, or create a *branch*, leaving the original data intact. Step (5) of the linguistic use case is a good example of that process. In the GIS scenario, the original hurricane track received by the NHC is used to create four derived tracks, which create new data sets (branches).

Ownership and Permissions: The owner of a data set decides who can read and/or write the data. If data are read-only, a private copy can still be created and changed. That process, however, must not pollute the original ownership and permissions. For example, the original hurricane track is owned by NHC. The UFL copies that track, and re-added four additional versions to the archive: these should have the same or less access permissions as the initial NHC track. In the linguistic use case, dependent tiers should share the same permissions as the tiers they depend upon, and should hence inherit these permissions.

3 Consistency and Scalability in Data Grids

The basic functionality of replica systems is simple: a replica catalog maintains a mapping between logical names (entities) and a set of distributed identical physical files (replicas). Often, entities are annotated with meta data.

In order to be scalable and performant in distributed environments, replica systems deploy a variety of latency hiding, data caching and other optimization techniques, which can make the implementation of a replica system non-trivial. The basic feature set described above is, however, easy to implement, and is representative for the majority of replica systems.

3.1 Consistency Considerations

The basic operations listed above have subtle implications for data consistency of replicas, but also for the consistency of the name space of the replica catalog. For example, as two remote users change two replicas of the same entry at the same time, the replicas will cease to be identical. Similarly, operations in the logical file hierarchy can result in inconsistencies.

The problem is somewhat simplified by the fact that name space and meta data are often held in a central (though often replicated) data bases [22]. Hence, consistency can be provided by insuring consistency of that data base, which is in itself a well understood and solved problem. Also, the replication of these data bases over a small and rather static set of services is well implementable [23,10].

Operations on the logical namespace are usually small and, in fact, often atomic. The same does not hold for operations on the replicas: the access patterns to the physical files are difficult to predict, and consist of potentially large numbers of small operations. The provision of consistency for the replicas therefore constitutes a significantly more difficult problem – see [22] for discussion.

3.2 Scalability Properties

Scalability issues of replica systems in respect to replica location management, selection, provisioning and transport are basically solved [10,9,12]. Data consistency, however, is expensive and impacts the scalability of these systems [22]. Todays grid replica system thus rarely provide consistency guarantees, and are often targeting on Write-Once-Read-Many use cases [10].

Scalability of ownership and permission enforcement are often provided by external systems, such as GAS [24] and GridShib [25]. These are known to scale well for large environments, allowing replica systems to profit from their scalability. It must be noted that the user level *management* of ownerships and permissions can be tedious. Domain specific interfaces are often required for that task [26]. We think that the scalability of ownership *management* breaks the shown use cases: the creation of derived data sets would require the intervention of the owner of the *original* data set, to approve or decline the derivations dependent on the data access policy intended for the *new* data set.

4 Data Versioning in Replica Systems

4.1 Consistency and Versioning

As discussed in the last section, consistency is usually considered to be a property of an instance of an entity which is under the management of the system: that entity is consistent if all its replicas are identical; it becomes inconsistent if one replica gets changed; and can become consistent again as these changes get propagated to all other replicas [22].

We want the reader to take the perspective of the end users: consistency then often means that any operation performed on any replica of an entry is reflected by any subsequent operation on any (same or other) replica of that entry.

For example, data written to a particular replica should be retrievable by a subsequent read operation on that entry. The entry would be inconsistent if that read operation would not return the new data, because it happened to get performed on a replica which is not yet in sync with the changed replica instance.

Now, from that perspective it would actually be simple to achieve consistency with the following scheme:

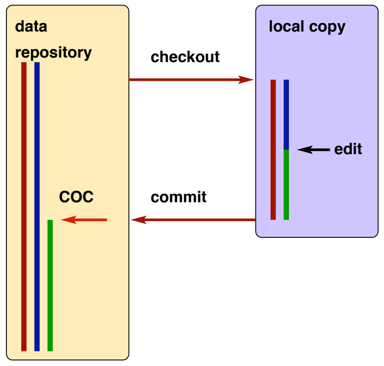

Fig. 2. Proposal: Changing a replicas results in a new version of its entity.

1. A replica systems entry `A` is tagged to have an initial version id `A-1`, and has 2 replicas `A-1a` and `A-1b`.
2. A end user intends to perform a write operation on `A`. On `open()`, the replica system performs a `copy()` of any replica to `A-2a`, which is not yet published in the replica catalog. The user edits that replica.
3. On completion of the edit, the replica `A-2a` is published as *new entry*, which
   – is a new version of `A`, named `A-2`
   – has only one replica (that is `A-2a`)
4. `A-2` gets lazily replicated for availability etc., and slowly spreads in the system (`A-2b`).
5. Any subsequent read operation on `A` is automatically rerouted to the *newest* version `A-2`.

With the above schema, `A` never gets into an inconsistent state. The scheme does introduce, however, the possibility conflicts when multiple users edit `A` at the same time. We will discuss that point below, and propose a solution.

The schema involves only operations which are very well supported by todays replica systems, and are known to perform well, and to be scalable:

– replica selection based on meta data (version)
– replication (local copy, spreading of new entry)
– change meta data
– publish new entry

The reader might feel familiar with the described scheme – in fact it is the very scheme which is used by concurrent versions systems such subversion [27]:

changes are performed on local copies of an entity, which is then synchronized with the central repository and with other, distributed copies. Changes *always* result in a completely new version of the entity – only performance optimizations lead to the more efficient exchange of differential updates.

The user causing the creation of a new version is the *owner* of that new version, as he created the new entity, and registered it to the system. For that, write access to the collection hosting the original entity is required. That change in ownership does not imply a change of permissions to other users – by default, the original access policy should stay in place. The new owner can however change access permissions. We don't see any violation of the security contract here, as the user could have softened these permissions by creating an unprotected private copy anyway, as he was allowed to create a copy in the first place. Further, the permissions for the *collection* does not change, so that the new access permissions cannot circumvent a more stringent access policy on that collection.

Trade-Offs and Optimizations

The approach to treat data edits implicitly as version bumps has two significant drawbacks. Firstly, the algorithm requires additional storage space as it increases the numbers of entities in the system on each edit, and all entity versions need to get replicated. Secondly, as mentioned above, it introduces conflicts when multiple users are editing replicas of the same entity.

It is tempting to shrug off the space trade-off as "insignificant in nowadays unlimited storage systems", but that would negate the communities these systems are targeting: large distributed VOs with extremely large data sets.

We think, however, that (a) the algorithm as-is fits those use cases which have only a limited and comparably small number of write operations, and that (b) very simple space optimizations can be applied. Those space optimizations can, for example, weed out versions which are superseded by newer versions, have never been used, and are not explicitly tagged for keeping (e.g. for auditing).

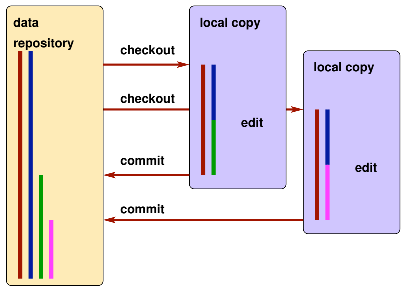

Fig. 3. Proposal: race conditions can effectively be avoided by modeling them as branching.

Resolving Conflicts

Conflict could of course be resolved manually. That however seems impractical for large binary data we are considering here. The conflicts could also be resolved by treating all new replicas not only as new versions, but as *branches*.

In th linguistic use case, branching is effectively what happens in step (5): two groups change the content of a collection, incompatibly, at the same time: branching is the natural way to cope with such operations.

5 Implications for Data Grid Implementations

The previous sections motivated the introduction of two basic capabilities, versioning and branching, on top of conventional replica systems. Both operations can be build upon a very small set of basic operations:

– attach meta data to entities (version/branch information)
– update meta data consistently
– replica selection based on these meta data
– replication (local copies, spreading of new entries)
– publish new entries

As discussed in Sect. 3, these operations are all available in replica systems and perform and scale well. The most critical operation is the meta data update, as it requires consistency management for meta data – as discussed above, that is well understood and implemented in typical replica systems [10,12,22,23].

Further, the implementation of versioning and branching operations does not need to be atomic. The initial step is, in both cases, the creation of a replica which is not registered in the replica catalog – which is an supported operation. After changing that replica, it needs to be registered as new entry, i.e. with a new set of meta data – again a supported operation. The implementation of versioning and branching can hence, in our opinion, be implemented on top of existing and deployed replica systems, in user space (although an implementation as system extension would allow for more efficient space and availability optimization, as described earlier).

6 Conclusion

We propose to extend existing replica systems with scalability enhancing features: (a) automatic entity versioning on replica changes, and (b) automated branching on colliding replica changes. Combined with sticky permission and ownership policies of entity collections, this would allow for

– performant consistency on concurrent data changes
– scalable permission maintenance
– interactive collaborative usage of data
– community shared data maintenance

The only trade-off we are aware off is an increase in storage space, depending on the number of writes, which can be relieved by optimizing storage policies.

These features should allow to apply grid replica techniques to large scale GIS environments, and would allow the GIS community to benefit from the positive effects replica systems showed in other scientific and commercial environments.

We are aware that this paper is rather silent on the specific details of ownership and permission management, and their application to the described use cases. We felt that this topic was not adequately to be handled in the limited space available, and thus will present that in a future publication.

7 Acknowledgments

The presented ideas base on many discussions with colleagues at the Max-Planck-Institute for Psycholinguistics Nijmegen, Netherlands, at the Center for Computation and Technology at the Louisiana State University, USA, and in the Data Area of the Open Grid Forum [28]. Part of this work was funded by the EU Project XtreemOS (IST-033576).

References

1. Paul Avery and Ian Foster. The GriPhyN Project: Towards Petascale Virtual Data Grids. *The 2000 NSF Information and Technology Research Program*, 2000.
2. A. Abramovici, W.E. Althouse, R.W.P. Drever, Y. Gursel, S. Kawamura, F.J. Raab, D. Shoemaker, L. Sievers, R.E. Spero, and K.S. Thorne. LIGO-The Laser Interferometer Gravitational-Wave Observatory. *Science*, 256(5055):325–333, 1992.
3. E. Deelman, C. Kesselman, G. Mehta, L. Meshkat, L. Pearlman, K. Blackburn, P. Ehrens, A. Lazzarini, R. Williams, and S. Koranda. GriPhyN and LIGO, building a virtual data Grid for gravitational wave scientists. *High Performance Distributed Computing, 2002. HPDC-11 2002. Proceedings. 11th IEEE International Symposium on*, pages 225–234, 2002.
4. B. Segal, L. Robertson, F. Gagliardi, F. Carminati, and G. CERN. Grid computing: the European Data Grid Project. *Nuclear Science Symposium Conference Record, 2000 IEEE*, 1:2, 2000.
5. A. Rajasekar, M. Wan, R. Moore, W. Schroeder, G. Kremenek, A. Jagatheesan, C. Cowart, B. Zhu, S.Y. Chen, and R. Olschanowsky. Storage Resource Broker-Managing Distributed Data in a Grid. *Computer Society of India Journal, Special Issue on SAN*, 33(4):42–54, 2003.
6. C. Baru, R. Moore, A. Rajasekar, and M. Wan. The SDSC Storage Resource Broker. *Proceedings of the 1998 Conference of the Centre for Advanced Studies on Collaborative Research*, 1998.
7. Reagan Moore and Andre Merzky. Persistent Archive Concepts. Technical report, Global Grid Forum, December 2003. GFD.26.
8. Heinz Stockinger, Omer F. Rana, Reagan Moore, and Andre Merzky. Data Management for Grid Environments. *Lecture Notes in Computer Science*, 2110:151–160, 2001.
9. A.L. Chervenak, N. Palavalli, S. Bharathi, C. Kesselman, and R. Schwartzkopf. Performance and Scalability of a Replica Location Service. *High Performance Distributed Computing Conference (HPDC-13), Honolulu, HI, June*, 2004.
10. A. Chervenak, B. Schwartzkopf, H. Stockinger, B. Tierney, E. Deelman, I. Foster, W. Hoschek, A. Iamnitchi, C. Kesselman, and M. Ripeanu. Giggle: a Framework for Constructing Scalable Replica Location Services. *Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, pages 1–17, 2002.
11. P. Kunszt, E. Laure, H. Stockinger, and K. Stockinger. Advanced Replica Management with Reptor. *5th International Conference on Parallel Processing and Applied Mathemetics, Sept*, 2003.
12. L. Guy, P. Kunszt, E. Laure, H. Stockinger, and K. Stockinger. Replica Management in Data Grids. *Global Grid Forum*, 5, 2002.
13. C. Lagoze, W. Arms, S. Gan, D. Hillmann, C. Ingram, D. Krafft, R. Marisa, J. Phipps, J. Saylor, C. Terrizzi, et al. Core Services in the Architecture of the National Digital Library for Science Education (NSDL). *Arxiv preprint cs.DL/0201025*, 2002.

14. A. Solomonides, R. McClatchey, M. Odeh, M. Brady, M. Mulet-Parada, D. Schottlander, and S.R. Amendolia. MammoGrid and eDiamond: Grids Applications in Mammogram Analysis. *Proceedings of the IADIS Intl. Conference: e-Society*, pages 1032–1033, 2003.
15. Kathrin Kirsch. Working Cross-Platform: a case Study in Coding, Sharing and Analyzing Corpora. Presentation at the 27th annual meeting of the DGfS 2006 in Bielefeld: Workshop on Language Archives - Standards, Creation and Access (AG-6), February 2006.
16. D. Broeder, F. Offenga, D. Willems, and P. Wittenburg. The IMDI Metadata Set, Its Tools and Accessible Linguistic Databases. *Proceedings of the IRCS Workshop on Linguistic Databases, Philadelphia*, pages 11–13, 2001.
17. Z.R. Peng and M.H. Tsou. *Internet GIS: Distributed Geographic Information Services for the Internet and Wireless Networks.* Wiley, 2003.
18. Editor: George Percivall. OGC Reference Model (ORM). Technical report, Open Geospatial Consortium, September 2003.
`http://www.opengeospatial.org/specs/?page=orm`.
19. A. Matheus. Authorization for digital rights management in the geospatial domain. *Proceedings of the 5th ACM workshop on Digital rights management*, pages 55–64, 2005.
20. A. Vckovski and A.J. Vckowski. *Interoperable and Distributed Processing in Gis.* CRC Press, 1998.
21. Gabrielle Allen, Philip Bogden, Gerald Creager, Chirag Dekate, Carola Jesch, Hartmut Kaiser, Jon MacLaren, Will Perrie, Gregory Stone, and Xiongping Zhang. GIS and Integrated Coastal Ocean Forecasting. *Concurrency and Computation: Practice and Experience*, 00(2), 2006.
22. D. Dullmann, W. Hoschek, J. Jaen-Martinez, B. Segal, A. Samar, H. Stockinger, and K. Stockinger. Models for Replica Synchronisation and Consistency in a Data Grid. *10th IEEE Symposium on High Performance and Distributed Computing (HPDC-10)*, 2001.
23. T. Anderson, Y. Breitbart, H.F. Korth, and A. Wool. Replication, consistency, and practicality: are these mutually exclusive? *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pages 484–495, 1998.
24. S. Cannon, S. Chan, D. Olson, C. Tull, V. Welch, and L. Pearlman. Using CAS to Manage Role-Based VO Sub-Groups. *Proceedings of Computing in High Energy Physics (CHEP'03)*, 2003.
25. T. Barton, J. Basney, T. Freeman, T. Scavo, F. Siebenlist, V. Welch, R. Ananthakrishnan, B. Baker, M. Goode, and K. Keahey. Identity Federation and Attribute-based Authorization through the Globus Toolkit, Shibboleth, GridShib, and MyProxy. *5th Annual PKI R&D Workshop, April*, 2006.
26. Open Grid Portals.
`http://www.opengridportals.org/space/Portlets/Security`.
27. B. Collins-Sussman, B.W. Fitzpatrick, and C.M. Pilato. *Version Control with Subversion.* O'Reilly, 2004.
28. Data Area of the Open Grid Forum (OGF).
`http://www.ggf.org/ggf_areas_data.htm`.