Armada: an evolving database system
Groffen, F.E.

Citation for published version (APA):

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: http://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.
Conclusions and Future Work

9.1 Contributions

As the introduction of this thesis mentions, trends in computing change. Driven by requirements, ideals and most of all humans, our creation changes, and will change many times in the future. This thesis builds upon a trend that puts focus on the distribution of tasks over multiple machines. Scaling of performance is no longer seen as a viable option for a single, large machine. Instead, each partial machine that the entire system is made of, adds its own share to the performance of the system.

Also database management systems have been following this trend. For them, data growth and the processing thereof are simply growing out of bounds of a single system. However, as growth in terms of data and usage continues, the number of systems grows as well, causing an inevitable administration hurdle. To overcome this hurdle, a system has to manage itself to a large extent, alleviating the database administrators. This thesis deals with the autonomy implied by this self management in a distributed setting, which is expected to evolve where necessary.

The aim of this thesis is to be an exploration of autonomy, decentralisation and evolution in the context of database systems. Previous research in these three areas is mainly focused on decentralisation with autonomy as side-effect. In particular P2P systems which are primarily decentralised have been given a lot of attention. Autonomy is considered to go hand in hand with decentralisation, since a central controlling component is absent in decentralised systems. However, the amount of autonomy given to the participants of distributed systems is limited. Evolution has mainly been approached from a system internal angle.

The contribution of Armada extends autonomy to the level where participants are initiators in the system. Through this initiative, evolution of the
The entire system is achieved. Evolution, here, is not just an internal matter, but at the level of the global system, and can be seen as self-managing behaviour. The proposed Armada model decentralises by storing partial catalog information throughout the entire system.

The autonomy of the system enforces clients to play an active role in query resolution. This gives the client an unusual task, for the benefit of having the user controlling the query process. This is in particular useful using the incremental query evaluation that is a result of the Armada lineage tree. There are natural points where the query can be stopped or suspended, since data is divided in blocks. These blocks are defined by arbitrary functions, allowing full control over the characteristics of the data in a block.

9.2 Research Questions

In our exploration towards autonomy, decentralisation and evolution of database systems, we were lead by a general research question, consisting of four more specific questions. We first answer the four questions, followed by the answer to the general question.

9.2.1 In what way can we distribute data in a dynamically evolving system using site local decisions and avoid global site control?

The Armada model from Chapter 3 describes how data can be spread over multiple sites starting from a single one. Growth in this model is achieved by performing operations which involve sites being added to the system. The operations are performed by the local site that was triggered to do so by e.g. a resource limitation. Such site is an initiator to resolve a local problem, on its own. Other sites in the system need not to express agreement, or be informed of the performed operation. Obviously, this means only those sites which are not directly involved in the operation. This independence of sites for their local decisions is due to the trail administration of the Armada model. The trails store pointers to other sites, that may or may not be up-to-date. Upon creation, history information is inherited from the local site, allowing each site to refer back to predecessors. Since the pointers may not be up-to-date, when following them it has to be accommodated for that they are out of date. This leads to locally performed operations to become visible when they are referenced. Since operations always redefine where data is that used to be on the original site,
and hence when following the trails, one never suddenly ends up in the wrong place.

The lineage trails of the Armada model are a way to administrate the whereabouts of data, without a central catalog scheme. Dynamically evolving systems benefit from this scheme, since they can expand when and where necessary without being hampered by costs of e.g. global updates and checking for their consistency. The use of arbitrary functions in operations, gives full control to the local site performing the operation.

9.2.2 What is the role of application clients in an autonomous, distributed database management system?

To maintain their high level of autonomy, servers do not perform work for which others are responsible. In practice this boils down to the clients in the system acting as directors in query execution. This unusual role for a client is discussed in Chapter 5. While it is intensive to perform all actions during query resolution, agents can be of help to the client. Agents can perform simple roles, such as following redirects and assembling a query result, if the client desires no explicit control over that process.

The process of query resolution can become expensive when an Armada grows large. Since clients, whether or not helped by agents, need to traverse the tree, the bigger it becomes, the more steps are possible. In Chapter 6 this process of localisation is further explored. To reduce the number of steps per query, the client or agent best uses a cache of visited trails for subsequent reuse.

9.2.3 How can incremental scalability become a natural component in an evolving system?

The operations from the Armada model described in Chapter 3 form the basis for a system to evolve. By means of these operations it can grow or shrink over time. This behaviour is already part of the design of the model and hence a natural component. The operations, however, need the functions to specify how exactly the operations need to be performed. In Chapter 4 we identified a number of functions and described their effect both separate as well as in combination with each other. In addition we looked at how functions can be chosen automatically for the situation at hand, aiming at a naturally self-evolving system.
9.2.4 To what extent are existing common techniques to manage a catalog sufficient to support autonomy, decentralisation and evolution?

The proposed model in this thesis breaks with traditional methods. Nevertheless we tried to simulate it using the SQL language in Chapter 7. While querying could work on existing database systems using this SQL approach, the autonomy considerations of the Armada model have to be ignored, since no client redirection is possible. The remote querying facilities are also not standardised and not necessarily sufficient to perform Armada operations. The SQL approach also falls apart with updates because of limitations of the used views.

However, while existing database systems are incapable of supporting Armada when implemented in the SQL language, it is not impossible to implement Armada. In Chapter 8 an approach on a deeper level towards the database kernel is explored. It shows that on those lower levels there is more freedom which allows to make a sufficient implementation. If one is willing to omit user interaction during query resolution, the interface can even use the SQL language, because the Armada implementation is below it, and invisible for the user.

9.2.5 How to support a continuously evolving database management system consisting of autonomous sites and a decentralised catalog?

In the previous sections we answered the four specific research questions. Now we are able to answer the main research question of this thesis.

Our exploration in the area of autonomous distributed database systems has resulted in a model which describes decentralised autonomous evolution in an environment of computers and schema-based data. The model handles the trade-off between full replication and centralisation of the catalog, by well targeted partial replication of the catalog meta information. This way, the catalog is distributed over the entire system as part of its own evolution process.

An important aspect of the model, is its assumption of local autonomy. It is this autonomy that allows the local sites to initiate a next step in the evolution of the system. But the autonomy also affects the users of the system. We have explored the effects of this autonomy on the clients in the system, in particular how well they can perform queries. We can conclude that it is possible to do so, even though not with standard techniques available today.

A key to the self-evolution of the system is in the way functions, as part of
the Armada model’s operations, are created. The functions define how the data is distributed, and when created by local sites based on local conditions, also form the driving power behind evolution of the system as a whole.

9.3 Future Work

With our exploration we have scratched the surface of a decentralised, autonomous and evolving world. We did not round the Cape, but fortunately did not ask to keep on trying until the youngest day either. Docked at the Table Bay, we leave a lot of work left for others to pick up and continue with. The following areas are of particular interest.

**Functions** The functions that back up the operations are only partially addressed in this thesis. Not only have we largely ignored the cloning and combining functions, but also the chunking functions have been marginally addressed with only a range function implementation. Different functions have different behaviours and they may be more beneficial under certain workloads.

In addition the conditions under which decisions for cloning and chunking are made, are a separate topic potentially borrowing from the artificial intelligence area when seen from a multi agent system angle.

**User Input** One of the interesting properties of the Armada model is the close user interaction. This allows for handling very specific user demands. We have left this mainly untouched, but we can think of applications where this control is beneficial. A few ideas are skipping certain boxes in query resolution, aborting the execution, or suspending it for a long time.

Performing such interaction requires changes to existing user interfaces though. To benefit optimally from this interaction the user needs to be provided with many meta data over the system and the state of the query resolution process.

**MAL Architecture** The approach described in Chapter 8 has been validated against the system in question. However, the missing parts to make it a working system, still need to be implemented. The described remote module, and the Merovingian service described in Appendix A are blocks for an actual implementation, but they need to be put into action still. The most work is in the optimiser architecture that plugs in the armada behaviour on top of an SQL
plan, as to make it work from a user’s perspective. Of course the previously mentioned user interaction considerations are also a possible design and implementation adventure.

**Physical Allocation** While we described how sites can be selected from a pool, the way in which this pool is allocated and maintained leaves some questions. The physical implementation of “plugging in machines” still requires some administration, and a machine that is removed should be properly unregistered by making sure its data is cloned to another machine.

The process of finding a new site to host a box on, is rarely a case of choosing one site from a pool of homogeneously typed machines. Instead, a mix of fast, slow, multi-way, large and small systems in terms of processing power and data storage capacity is more likely to be in the pool. Next to these properties, also the network link that connects the sites is an important factor in their performance. These properties ask for a careful cost/revenue analysis, based on predicted loads and functionalities. Some machines may even announce their life time in the system, making them only useful for e.g. facilitating a burst.