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.
In a world where data usage becomes more and more widespread, single system solutions are no longer adequate to meet the data requirements of today. No longer one monolithic system, but instead a group of smaller and cheaper ones have to manage the workload of the system, preferably as stable as the large single systems currently in use.

The ultimate goal is to have a self-managing and self-maintaining cluster of machines that just needs maintenance in terms of physically adding and removing hardware every once in a while, to cope with changed requirements. Close to this objective are Peer-to-Peer (P2P) systems, which are well-known on the internet, and quite effective in distributing data over the network. However, these systems typically distribute only certain data over the network as side-effect of certain user demands.

This thesis explores the landscape of self-managing database systems. It takes autonomy, decentralisation and evolution as starting point for this exploration. Autonomy of an individual system allows how much a system is able to control itself, and make decisions for itself that put it in a better position, for instance by temporarily refusing to do work for others. This self-regulation allows for evolution of the entire system, where individual components work towards a new structure of the system that better matches the current requirements. Such approach leads to decentralisation, as there is no hierarchy since all systems are autonomous.

At the heart of this thesis is the Armada model which describes a method to distribute relational data, as found in typical database systems, over a cluster of machines. The model takes autonomy, decentralisation and evolution as starting points, resulting in a distributed administration. Since the administration is not managed in a single location, this way local systems can use their autonomy and change the administration for the part they are responsible for. Each system can do this without harming any of the other systems, thereby supporting evolution. Because this gives each system a large degree of freedom, they can even choose how to perform for example a split of the data, using the right methods to reach the required goal, if they deem this necessary.

A consequence of having autonomous systems in a cluster is that users of the system have to face systems that refuse to do work on their behalf. This translates into an active client model, where clients are responsible for the execution of their own queries. This can be intensive and unfriendly for a human user. Fortunately it is possible to automate a lot of necessary work in an agent that works on behalf of the user, by communicating to the systems. However, this comes at the price that this way agents remove the possibility for the user to influence the execution process, such as stopping the execution after a review.
of intermediate results.

Agents that work on behalf of a user, looking for data in an Armada tree, need to hop around the cluster from system to system. The more hops an agent makes, the longer it takes, and hence the lower the performance of query execution. It is beneficial if the agent can reduce the number of steps it has to make, which it can do by caching information on the whereabouts of data it encounters when searching. The next time the agent needs to handle a query, it can then first consider its cache to see if it can directly go to the right site, or one nearby. In practice this allows an agent to quickly reduce the number of hops it has to make per query.

It is possible to map the Armada model to SQL, using views. This way, each individual data block can be represented by a view, that points to the right table, or when no longer existing, the replacement tables. This way an ordinary query can become a query over a large amount of tables through these views. While this works fine for expansion of the database, as well as querying the data within it, updating or inserting data is a problem, since the current SQL implementations do not, or not sufficiently, support updates on views, which in the Armada case can be complicated. Hence this approach turns out to be of limited use.

To solve the above problem, an Armada implementation deeper into the database system is necessary, such as at the MAL level of the MonetDB database. On this level, which is directly on top of the core engine, there are many degrees of freedom that allow to do more complex operations and optimisations. On this MAL level, an Armada system that supports reads and writes can be implemented.