ICTTE International Conference on Technics, Technologies and Education ICTTE 2014
October 30-31 2014, Yambol, Bulgaria
https://sites.google.com/a/trakia-uni.bg/ictte-2014/

SINGLE-CLICK TO DATA INSIGHTS TRANSACTION REPLICATION AND DEPLOYMENT AUTOMATION MADE SIMPLE FOR THE CLOUD AGE

Dimitar G. Nedev[1], Niels Nes[2], Hannes Mühleisen[2], Ying Zhang[1], Martin Kersten[2]
[1]MonetDB Solutions
Science Park 123, 1098XG Amsterdam, Netherlands
e-mail: Dimitar.Nedev@monetdbsolutions.com, Ying.Zhang@monetdbsolutions.com
[2]CWI
Science Park 123, 1098XG Amsterdam, Netherlands
e-mail: Niels.Nes@cwi.nl, Hannes.Muehleisen@cwi.nl, Martin.Kersten@cwi.nl

Abstract: *In this report we present out initial work on making the MonetDB column-store analytical database ready for Cloud deployment. As we stand in the new space between research and industry we have tried to combine approaches from both worlds. We provide details how we utilize modern technologies and tools for automating building of virtual machine image for Cloud, datacentre and desktop use. We also explain our solution to asynchronous transaction replication MonetDB. The report concludes with how this all ties together with our efforts to make MonetDB ready for the age where high-performance data analytics is available in a single-click.*
Keywords: *databases, cloud computing, deployment automation, transaction replication.*

1. INTRODUCTION

Cloud computing has become one of the leading utilities for data processing. Cloud service helps both the academic researcher and commercial organizations in getting more out data with fewer resources. In just a few minutes one can launch a high-performance virtual instance running in the cloud and shut it down once its now longer needed [10].
The MonetDB column-oriented store has a long track record in business analytics and it is often used as a scientific database, managing large volumes of data. A cloud deployment of MonetDB will give its users the fast query responses of in-memory optimized database, while maintaining the convenience of immediate access to compute resources.
Cloud computing and analytical database systems are not only useful to research institutions, but to enterprises as well. As such, adapting the cloud deployment of MonetDB to support industrial customers will support a wider range of users easy access to a high-performance database. In our cloud integration effort we focus on both making the process of using MonetDB effortless, as well as making it an appealing tool to both academic and industry users.
Our design extensions to the database system focuses on long-term maintainability. We achieve this by re-using the existing facilities in both MonetDB and third party software, avoiding major redesigns. For our cloud integration we have opted for a toolset that allows us to automate the process of creating a virtual image for multiple platforms. With a 'write once use many' approach we cover a large range of cloud service providers with a single set of deployment templates and scripts. Similarly, the database replication technology of MonetDB uses as much as possible its proven transaction logging subsystem. With transaction replication support, we have added a high-availability feature to MonetDB.

Deployment automation and transaction replication together make MonetDB more appealing to a larger set of users. Next the ease of use deployment, high availability is a critical requirement for commercial database users. Downtime can negatively impact both researcher and industrial user on database systems. With this work we aim to bring MonetDB in the Cloud age, giving data analysts a powerful tool that is only a single-click away.

2. CLOUD DATABASE DEPLOYMENT

Cloud computing significantly shortens the time it takes to bring an idea from inception to implementation. While previously one had to wait for authorization or even procurement of the compute resources, nowadays accessing high-performance is a click away [10]. Such a convenience was previously only available if one had direct access to the hardware and software required to do the job. In the case of large-scale data analytics a more powerful machine would have been required, thus more resources to be spent on their procurement. Buying hardware generally falls under capital expenses, which are planned a year in advance. In other words the decision to buy new hardware for a project was made well ahead of time. As a result the machines are ordered at potentially higher prices, since hardware prices are constantly decreasing, as new and better performing models are released. In comparison, money spent on cloud instances generally fall under operational expenses, which can be allocated at much shorter periods. Additionally, underutilization of the hardware is a waste on both the initial investment and electricity. In contrast, cloud-computing resources can be used when needed and then shut down [3]. Given this trend, one can consider Cloud resources as of a common utility of the 21st century. In that is respect, cloud service providers are comparable to gas, electricity and telephone companies [4].
As mentioned, access to cloud resources, even very high-performance such, has never been easier. This is especially relevant for work that involves data processing. In both academic research and commercial companies, data analysis is often done in batches. Large dataset are processed, the output is analysed and presented. Once new data arrives, a new dataset is again processed. In most cases newer data is appended, with limited modifications to the previously stored information. For data intensive tasks, database management systems (DBMS) are a natural choice. As such, deploying an analytical DBMS in the Cloud is powerful tool for data processing and analysis.

3. COLUMN STORES AND MONETDB

MonetDB is an open-source DBMS for high-performance applications in data analysis, business intelligence, OLAP, GIS and data warehousing. These applications are characterized by very large databases, which are mostly queried to provide business intelligence or decision support. Similar applications also appear frequently in the area of e-science, where results from experiments are stored in a scalable system for subsequent scientific analysis.
The design of MonetDB is built around the concept of bulk processing: simple operations applied to large volumes of data make efficient use of the hardware for large-scale data processing. This focus on bulk processing is reflected at all levels of the architecture and the functionality offered to the user. MonetDB achieves its goal by innovations at all layers of a DBMS, e.g., a storage model based on vertical fragmentation (column store), modern CPU-optimized query execution architecture, automatic and self-tuning indexes, and run-time query optimization [7].

MonetDB stores data in columnar format and the columns are virtually stitched together to form tables (Figure 1**Figure 1**). This is one of the main differences between MonetDB and the common database systems, which store data in rows. The vertical storage given column-stores some advantage in read-intensive workloads, such as data analytics. At the same time MonetDB is a fully functional relational database. It provides an SQL:2003 compatible

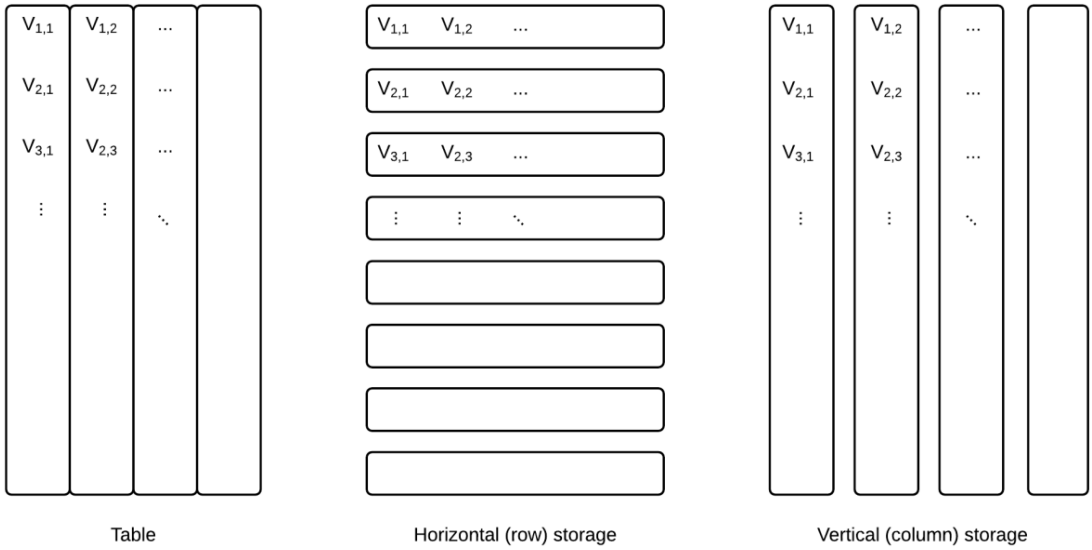

Figure 1. Row vs. Column oriented storage

query layer and connectors for most programming languages. This means that the common tools and applications designed for working with SQL systems will easily integrate with database system. In addition, MonetDB makes use of a number of novel techniques for efficient support of a priori unknown or rapidly changing workloads [7]. “Recycling” is a fine-grained flexible intermediate result caching technique [8]. “Database cracking” is an adaptive incremental indexing technique [17].

To further improve data analysis capabilities, MonetDB is also being integrated with the R statistical analysis software [11]. This will work both ways: a connection to database can be established from R and functions written in R can be evaluated directly in MonetDB. Statistical packages are optimized for advanced algorithms, while database systems provide fast access to large volumes of data. Their combination creates a powerful platform to speedup data discovery. The MonetDB.R connector decides which portions of the data analysis should be performed by either statically software or the database. This way each systems is used to its best performance. The user is freed from the tedious task to shuffle data around. This also significantly reduces the overhead of shuffling data between different systems, improving data processing times [9].

4. DEPLOYMENT AUTOMATION

In the MonetDB cloud integration we focused on facilitating easy deployment of ready-made MonetDB virtual images. The main idea is to make MonetDB easy to run and operate in both public and private clouds environments, as well as creating small enough images for desktop users to test. In other words, support faster deployment of MonetDB in any environment with a single click.

The main issue for such effort is the lack of coherent standard for a cloud platform or virtual machine (VM) images. OpenStack is an initiative to provide a standard open-source Infrastructure-as-a-Service (IaaS) platform, and while it has many industry supporters, most of the big public cloud providers do not support it [13]. Rather, the OpenStack APIs closely mirrors the Amazon Web Service (AWS) EC2 and S3 APIs [12]**.** As a result, deployments designed to work with OpenStack can be ported to AWS with limited effort. The importance of AWS compatibility (or portability) stems from the AWS position as an undisputed leader as a cloud service provider; both in terms of computing power availability and richness of available services [1]. While the emerging standard for VMs for commercial deployment is based on the Docker container concept, the big cloud service providers have just started supporting Docker [5].

This makes exclusive selection of the either OpenStack or Docker as the primary platform for VM image format not wise for the moment. We also observe a lot of development in the cloud services area and new standards or new adopters of existing APIs appearing quite frequently. Given these issues and trends, at the beginning of the project we set out to find a solution that allows us to easily maintain the VM images for most platforms with as little effort as possible.

We have chosen to use Packer - a tool for automating the VM image generation [15]**.** Packer is a versatile tool that supports most VM image formats including: Amazon Machine Images (AMIs) for AWS, Docker containers, VirtualBox OVF images, Google Compute Engine (GCE) images, OpenStack images, etc. The definition of an image generation is provided in a JSON templates and Packer can execute provisioning scripts before, during and after the process. In addition, Packer supports most common cloud configuration management tools such as: Chef, Puppet and Salt, providing end users a powerful array of enterprise-level tools for instance management [16]**.**

Since our implementation focused on maintainability of the VMs, we used a more software engineering process for writing our Packer templates and provisioning scripts, unlike the more system administrator approach of ad-hoc scripting. As such, the templates and scripts were designed to be highly reusable, with little to no code duplication.

The templates cover the VM image generation for common operating systems used in public and private cloud deployments: CentOS 6.5 (upgraded to version 7) and Amazon Linux 2014.03.1 (upgraded to version 2014.09). The Packer templates were initially configured to generate VirtualBox OVF images for desktop testing and AWS AMIs. We eventually added Docker containers as well. With these three VM image formats we covered the three critical user groups of MonetDB:

- VirtualBox OVF for desktop users, to try MonetDB on their workstations, before moving to a full-scale deployment on more powerful machines.
- AWS AMIs for public cloud users, looking for a ready-made MonetDB deployment, so that they can start with a single click in the AWS cloud.
- Docker containers for private cloud/datacenter, users who are looking to build their own infrastructure with MonetDB, based on existing hardware and their own virtualization resources.

The implementation methodology and configuration scripts re-usability enables creation of new VM images with very limited effort. Over the course of several months, new versions of the initially used operating systems were released and we upgraded our temples and scripts to support the new release. This was done in a straightforward and easy way, verifying our choice in methodology and technology.

The provisioning/configuration scripts were cut into small parts, each executing a specific part of the operating system configuration during the image building process. Like the templates, the scripts were also designed to work on as many operating systems (and OS versions) as possible. The provisioning scripts are meant to be simple and highly reusable –

being the building blocks of our implementation. The major actions performed by the scripts are:

- Unattended installation of the OS (which is needed for creating some image formats):
- Operating systems initial setup
- Adding software repositories
- Software installation
- Adding (unprivileged) users and groups for process sandboxing
- Software and services configuration
- Additional image/OS specific software installation
- Clean-up

The installed and configured software always includes:

- Base OS - for fast boot times and lightweight containers
- MonetDB with SQL and GEOM/GIS modules
- R statistical software, configured with MonetDB.R for direct data access

The end results are a number of small and easy to maintain templates and scripts that can be automatically executed and generate up-to-date virtual images with MonetDB and R. Publication to the public cloud or another repository can also be automated as Packer post-processor actions.

5. TRANSACTION REPLICATION

High-availability is critical for production environments. Downtime in an academic research organization can cause one to miss important deadline or delays in research results. Availability of mission critical commercial systems is of even higher importance. If the data warehouse goes down, neither business reporting nor planning can be done. High-availability is often achieved via redundancy of critical components. For database management systems this is achieved using multiple instances of the database running in parallel. The data is continuously synchronized between the instances. This ensures that in case a single server is lost, the other replicas can still serve queries.

To increase the availability of MonetDB servers, we have extended the system with support for transaction replication. This is achieved via log shipping (of the transaction logs) of a master instance to a number of slave instances. It is generally described as a pull model [14], where each slave pulls transactions independently and asynchronously. There is no master-slave information exchange (aside from transactions). Formally this is considered Lazy Centralized replication with Limited Transparency [14].

By default the MonetDB kernel stores transactions in Write-Ahead Log (WAL) files, before the transaction are persisted in the primary persistent storage. During the database start up, the transaction log files are read and any data changes non-persisted in the primary storage are applied. A MonetDB slave instance can be configured to read the WAL files of a master instance, load the transactions and persist the data in its own persistent storage.

On the master instance, the MonetDB should be configured to keep all transaction log files, even those for transactions already persisted in the primary storage. By default, the database cleans- up persisted transaction log files. The transaction log files on the master have to be shipped to the slave instance(s), which can be done using a highly available shared filesystem or alternative means.

On a slave instance, the location of the master transaction log files must be configured. In addition, the transaction drift threshold between the slave and the master must be set. The drift is the difference between the transactions processed by the master and the slave. If a slave detects that it has passed a pre-set threshold, it will not process any additional client

read queries until it catches up with the master. A slave must also be set to run in read-only mode.
The master is the only instance that can apply changes to the data (e.g. create, update, delete) to avoid any data inconsistencies. As such all slave instances must run in read-only mode, where data changes will be propagated only through the transaction-replication mechanism.
There are two possible cluster configurations, each with its pros and cons:

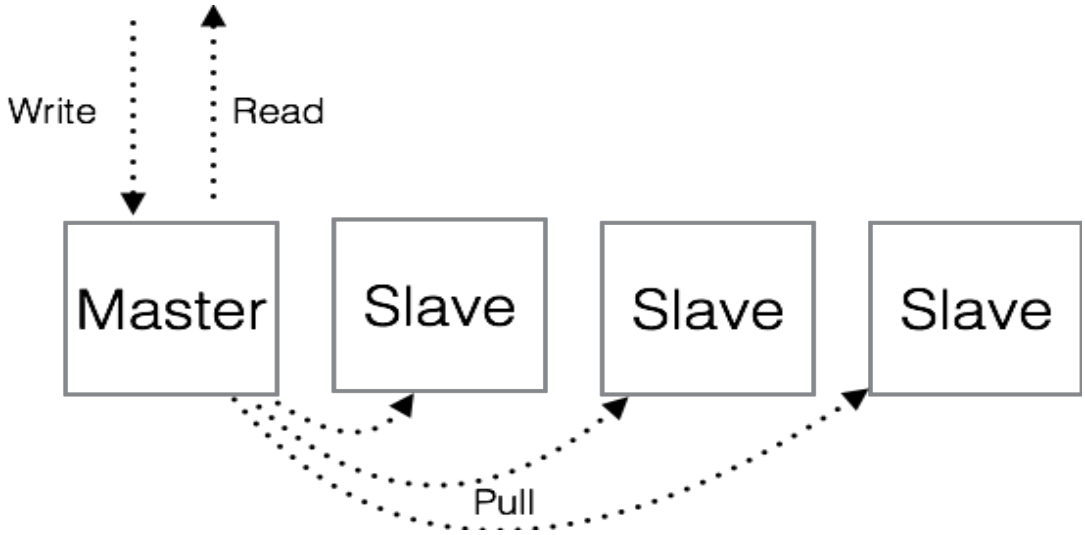

Figure 2. MonetDB warm standby configuration

- Warm standby slaves (Figure 2)
  - Single master instance, which can do read and write operations.
  - One or more slave instances processing no queries, only replicate the master transactions asynchronously.
  - Upon master failure, a slave instance can be restarted in non-read-only mode, to take the role of a master.

The warm standby configuration has the advantage that it can provide increased fault tolerance and is relatively simple to setup. Since the instance loading the data can be queried only, the queries operate on an always up-to-data store. The most significant disadvantage is the somewhat wasteful use of resources, as the slaves will to do any query processing

- Active-active with read-only slaves (Figure 3)
  - Single master instance, which can do read and write operations
  - One or more slave instances that can do read operations, next to replicating the master transactions asynchronously
    - These read-only instances can improve the read query load
    - Load-balancing must be provided on the client side
  - Upon master failure, a slave instance can be restarted in non-read-only mode, to take the role of a master.

The active-active configuration provides improved query capacity, compared to the single read instance warm standby. At the same time, queries sent to the slave instances can be executed on not-up-to-date data, since the transaction replication is asynchronous. This setup also comes at the price of increased complexity at the client side, due to the load balancing required for query distribution.

MonetDB is primarily designed as an analytical database. As such, data is best loaded in large bulk transactions. This will also guarantee that only single large files are shipped to the slaves for replication, minimizing the transaction drift.

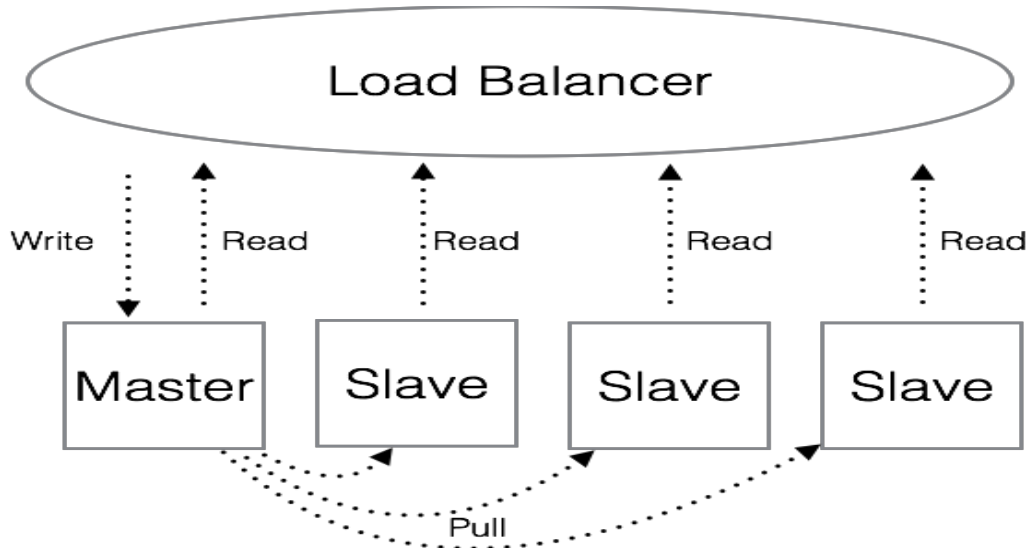

Figure 3. MonetDB active-active with read-only slaves

To set-up MonetDB transaction replication, first and foremost, the shipping of the transaction log files between the master and the slave instance(s) must be configured. The most straightforward way is to use a shared filesystem that can also provide high-availability for the master transaction log files. It is advisable to choose a shared filesystem that is also fault tolerant, such that loss of the master MonetDB instance will not lead to loss of the master transaction logs as well. This way the shared filesystem provides both log shipping, as well as log backup. Since the master instance preserves all transaction log files, there will be a complete copy of the database (in form of the transactions).
Good examples of such filesystem include Ceph and GlusterFS [6]. For public cloud deployment, the native storage of the cloud provider can be used. For example, in AWS one can setup WAL file replication to Amazon S3. S3 has high durability and availability [2], making it ideal for both log shipping and backup. If the MonetDB transaction log is written on S3, slave instances can read the files directly from the remote store over HTTP or mounted locally in user space. At the same time the high durability guarantees constant backup of the data. The main drawback is that S3 is object based [2], and on file update, the complete file must be uploaded. In the case of MonetDB, as the older transaction log files are not modified, the effect will be minimal. In addition, if the data is loaded in large bulk transactions, the impact is further minimized, since there will not be many files.
For the implementation of the transaction replication support we decided to take the path with least risks. First we evaluated the existing transactions logging scheme. On a slave instance, if the correct flags are set, the database will set up a second, read-only logger. Every few seconds the logger will scan its own (shared) directory and if new transactions are detected, it will load them and persist them in the local storage. If new tables or schemas were created in that transaction, these changes will not be visible immediately. To fix it, we also party reload the SQL store, forcing it to update the schemas and tables.

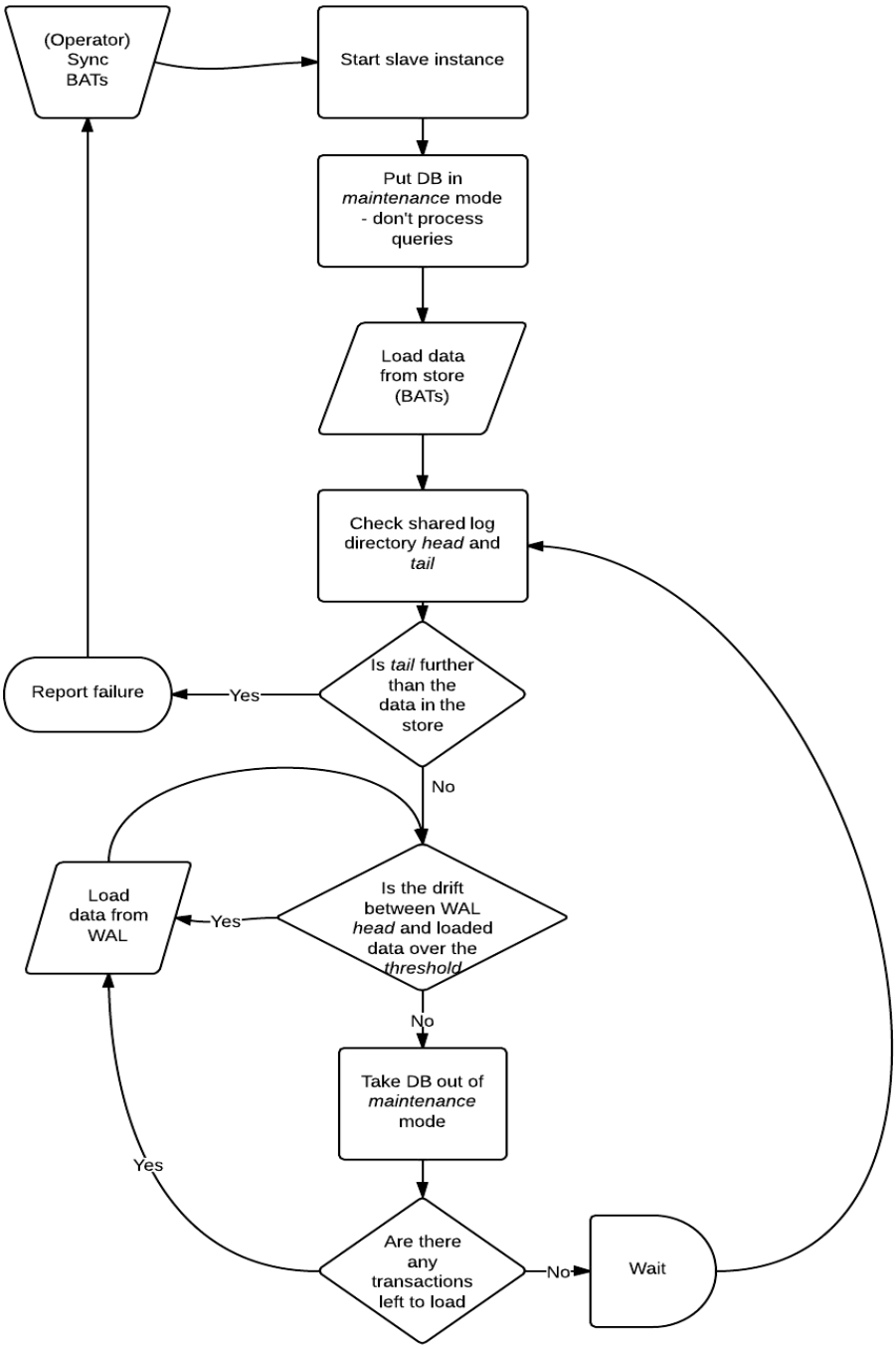

Figure 4. MonetDB transactions processing on a slave instance

The mode of operation of a slave instance is shown on Figure 4. The instance will start up in maintenance mode, not processing any queries. Any data in the slave's own WAL will be persisted first. Next the WAL synced from the master will be examined. The salve will verify if

it can replicate all data. In case the tail of master data is further than the head of the transactions on the slave, the instance will report failure. An operator/DBA must manually sync the instance coping the persisted data from the master to the slave. If tail of the master transaction is behind the head of the salve, the slave will then verify if the drift between the two instances is past the threshold. If that is the case, the slave will load all needed transaction from the synced master WAL. Once it's passed the threshold, it will unlock the store and begin to process read queries. At regular time interval the synced master WAL will be re-examined and the replication process begins anew.
On the master instance, as little as possible changes were done, in order not to compromise the performance of the database. The only major change is that more than one transaction is now preserved. This is needed since there is no communication between the master and the slave instance and all transactions that have to be replicated at the slaves need to be available.

6. CONCLUSION

Database systems running in the cloud have significantly reduced the time to process large volumes of data. To support this process database management systems have also adapted to the paradigm of the cloud age. As part of the cloud integration of MonetDB, we focused on the activities that serve the needs of data analysis best. Our deployment automation process ensures that the process of starting a new virtual instance with MonetDB is always simple. Pre-configured versions of MonetDB and R are ready to launch on demand. In addition, the methodology and technologies used minimize the maintenance effort and make upgrade simple. The transaction replication support, on the other hand, guarantees high-availability of running the MonetDB instances. The log shipping based approach reuses as much as possible the existing system. This way the risk and time to market of this solution is minimized. As a result, it now takes only a few clicks to start a new cloud instance of get insights from your data.

7. ACKNOWLEDGEMENTS

The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement N° 611068 (CoherentPaaS) and the COMMIT/ project.

8. REFERENCES

[1] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., in drugi. (2010). A view of cloud computing. *Communications of the ACM , 53* (4), 50-58.
[2] *AWS S3 FAQs*. (2014). Prevzeto 22. October 2014 iz Amazon Web Services: https://aws.amazon.com/s3/faqs/.
[3] Beloglazov, A., & Buyya, R. (2010). Energy Efficient Resource Management in Virtualized Cloud Data Centers. *10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid)* (str. 826-831). Melbourne: IEEE.
[4] Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., & Brandic, I. (June 2009). Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems , 25* (6), str. 599-616.
[5] *Docker 1.0 Linux Container System Takes Big Step*. (6. September 2014). Prevzeto 22. October 2014 iz InformationWeek: http://www.informationweek.com/cloud/infrastructure-as-a-service/docker-10-linux-container-system-takes-big-step/d/d-id/1269519.

[6] *File System Ceph*. (2014). Prevzeto 22. October 2014 iz Ceph: http://ceph.com/ceph-storage/file-system/.
[7] Ideros, S., Groffen, F., Nes, N., Manegold, S., Mullender, K., & Kersten, M. (2012). MonetDB: Two Decades of Research in Column-oriented Database Architectures. *IEEE Data Engineering Bulletin , 35*, 40-45.
[8] Idreos, S., Kersten, M. L., & Manegold, S. (2007). Database cracking. *Proceedings of the Biennial Conference on Innovative Data Systems Research (CIDR)* , str. 68-78.
[9] Mühleisen, H., & Lumley, T. (2013). Best of Both Worlds: Relational Databases and Statistics. *Proceedings of the 25th International Conference on Scientific and Statistical Database Management* (str. 1-4). ACM.
[10] Marston, S., Li, Z., Bandyopadhyay, S., Zhang, J., & Ghalsasi, A. (2011). Cloud computing - The business perspective. *Decision Support Systems , 51* (1), 176-189.
[11] *MonetDB to R Connector*. (2013). Prevzeto 22. October 2014 iz The R Project for Statistical Computing: http://monetr.r-forge.r-project.org.
[12] *Nova/APIFeatureComparison*. (2013). Prevzeto 22. October 2014 iz OpenStack: https://wiki.openstack.org/wiki/Nova/APIFeatureComparison.
[13] *OpenStack's Future Depends on Embracing Amazon. Now.* (24. July 2013). Prevzeto 22. October 2014 iz Cloudscaling: http://www.cloudscaling.com/blog/cloud-computing/openstack-aws/.
[14] Ozsu, M. T. (2007). *Principles of Distributed Database Systems.* (3rd, Ured.) Upper Saddle River, New Jersey, United States of America: Prentice Hall Press.
[15] *Packer*. (2014). Prevzeto 22. October 2014 iz http://www.packer.io.
[16] *Packer Dcoumentation*. (2014). Prevzeto 22. October 2014 iz Packer: http://www.packer.io.
[17] Sidirourgos, L., & Kersten, M. (2013). Column Imprints: A Secondary Index Structure. *Proceedings of ACM SIGMOD International Conference on Management of Data 2013 (SIGMOD)*.
[18] *Write once, read everywhere*. (2014). Prevzeto 22. October 2014 iz Gluster: http://www.gluster.org.