



## ACTiCLOUD: ACTivating resource efficiency and large databases in the CLOUD

Project No: 732366

H2020-ICT-2016-1

### D3.7: MonetDB ACTiCLOUD extensions v2.0

<b>Due date of deliverable:</b>	<b>M32 (2019/08/31)</b>
Actual submission date:	M32 (2019/09/03)

#### Executive summary:

This deliverable is a follow-up of D3.3 “MonetDB ACTiCLOUD Extensions v1.0”, in which we have reported the initial implementation of three MonetDB extensions to exploit the functionality enabled by the ACTiCLOUD architecture: i) *MALCOM* estimates the memory footprint MonetDB needs for a query before its execution; ii) *MonetDBLite-Java*, an embedded version of MonetDB for Java; and iii) *distributed query processing* features of MonetDB. In the second period of this project, we have both continued developing the aforementioned software, as well as developing new software. **In this deliverable**, we present one new MonetDB artefact and give updates to two existing artefacts.

*MDBconductor* is a new MonetDB artefact. It manages a cluster of MonetDB VM instances in the cloud, and uses the estimation of *MALCOM* to choose a suitable instance to execute a given query. It also takes care of transparently resizing an instance (e.g. memory and CPU cores) before executing a query, if none of the existing instances is suitable for certain query workload.

*MonetDBLite-Java* has been ported to the ARMv8 (AArch64) architecture. Furthermore, to ensure we can keep developing and maintaining this MonetDB artefact even after the end of ACTiCLOUD, we have done two major code refactoring. First, we have merged the *MonetDBLite* and *MonetDB* codebases into one codebase. *MonetDBLite* is the core library of *MonetDBLite-Java*. This merge also enables us to backport features, such as the in-memory only mode, from *MonetDBLite* to *MonetDB*. Second, we have migrated the MonetDB build-install-test system from the classic Autotools to the more modern CMake.

To allow MonetDB to scale out to a large number of instances, we have extended MonetDB’s *distributed query processing feature* to support predicate-based data partitioning and distributed updates.

**List of authors:**

Author	Affiliation
Pedro Ferreira	MDBS
Joeri van Ruth	MDBS
Panagiotis Koutsourakis	MDBS
Ying Zhang	MDBS

<b>Dissemination Level</b>	<input checked="" type="checkbox"/>	<b>PU (Public)</b>
	<input type="checkbox"/>	PP (Restricted to other programme participants)
	<input type="checkbox"/>	RE (Restricted to a group specified by the consortium)
	<input type="checkbox"/>	CO (Confidential, only for members of the consortium)
	Where restricted, access granted to:	
<b>Nature</b>	<input type="checkbox"/>	R (Report)
	<input type="checkbox"/>	P (Prototype)
	<input type="checkbox"/>	D (Demonstrator)
	<input checked="" type="checkbox"/>	<b>O (Other)</b>

<b>Review Status</b>	<input checked="" type="checkbox"/>	<b>Draft</b>
	<input type="checkbox"/>	WP Leader accepted
	<input type="checkbox"/>	QA approved
	<input type="checkbox"/>	Coordinator accepted

**Revision History:**

Version	Author(s) (Affiliation)	Notes
0.1	Ying Zhang (MDBS)	Initial ToC
0.5	Pedro Ferreira, Panagiotis Koutsourakis, Joeri van Ruth, Ying Zhang (MDBS)	First draft
0.7	Ying Zhang (MDBS)	Second draft
0.8	Christos Kotselidis (UNIMAN), Vasileios Karakostas (ICCS)	Reviewers
0.9	Pedro Ferreira, Panagiotis Koutsourakis, Ying Zhang (MDBS)	Revised after review
1.0	Ying Zhang (MDBS)	Final version

**ACTiCLOUD Consortium:**

Participant No	Participant organisation name	Short name	Country
1 (Coordinator)	Institute of Communication and Computer Systems	ICCS	Greece
2	Numascale AS	NSCALE	Norway
3	Kaleao Limited	KALEAO	UK
4	OnApp Limited	ONAPP	Gibraltar
5	University of Manchester	UNIMAN	UK
6	MonetDB Solutions B.V.	MDBS	Netherlands
7	Neo Technology	NEO	Sweden
8	UMEA University	UMU	Sweden



NUMASCALE



**Confidentiality:**

This document contains proprietary and confidential material of certain ACTiCLOUD contractors, and may not be reproduced, copied, or disclosed without appropriate permission. The commercial use of any information contained in this document may require a license from the proprietor of that information.

THIS DOCUMENT IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES

(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS DOCUMENT, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

**Table of Contents**

<b>1</b>	<b>Introduction .....</b>	<b>7</b>
1.1	Relevance to ACTiCLOUD’s Objectives, Use Cases and Business Scenarios .....	8
<b>2</b>	<b>MonetDB in the (ACTi)CLOUD .....</b>	<b>10</b>
2.1	MALCOM recap.....	10
2.2	Database server conductor in the cloud .....	11
<b>3</b>	<b>Updates of D3.3.....</b>	<b>17</b>
3.1	MonetDBLite-Java.....	17
3.2	Distributed Query Processing.....	19
<b>4</b>	<b>Conclusion .....</b>	<b>24</b>
<b>Appendix I References .....</b>		<b>25</b>
	Software repositories.....	25
	Software documentations .....	25

## Figures

Figure 1: ACTiCLOUD architecture: position of WP3 marked by the red box. ....	7
Figure 2: MALCOM workflow .....	11
Figure 3: Common workload patterns of business analytics applications. ....	12
Figure 4: MDBconductor architecture and workflow .....	13
Figure 5: MonetDB versus MonetDBLite codebases .....	18
Figure 6: query distributed data in a MonetDB cluster: tables <i>s1</i> , <i>s2</i> , <i>s3</i> , <i>s4</i> are exact copies of each other (shown in the same colour). This fact can be indicated using <code>CREATE REPLICATION TABLE</code> , so that when the table <i>S</i> is queried, the query will be run using the nearest replica. Tables <i>t1</i> , <i>t2</i> , <i>t3</i> , <i>t4</i> contain data partitions of a bigger table (shown in different colours). One can merge a number of partitions using <code>CREATE MERGE TABLE</code> , so that when the table <i>T</i> is queried, the query will be run on all partition tables included in <i>T</i> . ....	20

## Tables

Table 1: List of T3.3 software deliverables and relation with ACTiCLOUD's strategic objectives, use cases and business scenarios. ....	9
Table 2: a JSON object containing one heartbeat event.....	15

# 1 Introduction

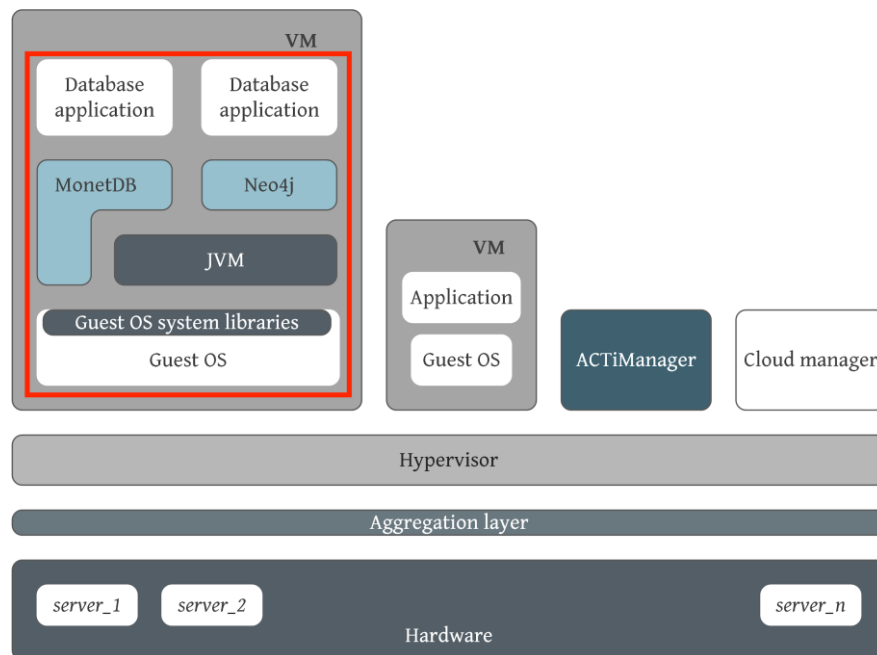


Figure 1: ACTiCLOUD architecture: position of WP3 marked by the red box.

**The main goal of ACTiCLOUD** is to develop a novel cloud architecture that will break the existing scale-up and share-nothing barriers and enable the holistic management of physical resources both at the local cloud site and the distributed levels, targeting drastically improved utilisation and scalability of resources.

In the ACTiCLOUD architecture, **WP3 “Evolution of Databases”** [M1-32] is positioned in the top layer as shown in Figure 1. Its main tasks are about extending/improving system libraries, JVM and different types of databases to make full use of the resource capabilities of ACTiCLOUD-enabled systems. **In Task 3.3 “In-memory databases”** [M4 - M32] we focus on extending the columnar database MonetDB towards ACTiCLOUD-enabled data centres.

**This deliverable** is a follow-up of D3.3<sup>1</sup>, in which we have reported the initial implementation of three MonetDB extensions to exploit the functionality enabled by the ACTiCLOUD architecture: i) *MALCOM*, estimates the memory footprint of MonetDB needed for a query before its execution; ii) *MonetDBLite-Java*, an embedded version of MonetDB for Java. It enables Java application developers to run MonetDB servers inside a JVM process and manage database servers directly in their Java code; and iii) *distributed query processing*, a master-worker architecture which transparently executes an SQL query on multiple MonetDB database servers. In the second period of the ACTiCLOUD project, we have both continued working on the software developed in the first period,

<sup>1</sup> P. Ferreira, P. Katsogridakis, P. Koutsourakis, J. van Ruth, and Y. Zhang. “D3.3 MonetDB ACTiCLOUD Extensions v1.0,” in ACTiCLOUD deliverables, Jun, 2018. Submitted in M18, evaluated during ACTiCLOUD P1 review. Available from <https://acticloud.eu/dissemination/deliverables>.

as well as developing new software. In this deliverable, we present a new MonetDB artefact and give updates to two existing artefacts:

- **MDBconductor:** is a new MonetDB artefact. It manages a cluster of MonetDB VM instances in the cloud, and uses the estimations of MALCOM to choose a suitable instance to execute a given query. It also takes care of transparently resizing an instance (e.g. memory and CPU cores) before executing a query, if none of the existing instances is suitable for certain query workload. In Section 2, we describe MDBconductor and the ongoing integration with the ACTiManager.
- **MonetDBLite-Java:** has been ported to the ARMv8 (AArch64) architecture. Furthermore, to ensure we can keep developing and maintaining this MonetDB artefact even after the end of ACTiCLOUD, we have done two major code refactoring. First, we have merged the MonetDBLite and MonetDB codebases into one codebase. MonetDBLite is the core library of MonetDBLite-Java. This merge also enables us to backport features, such as the in-memory only mode, from MonetDBLite to MonetDB. Second, we have migrated the MonetDB build-install-test system from the classic Autotools to the more modern CMake. In Section 3.1, we report the progress and our release plan for this work.
- **Distributed query processing:** to allow MonetDB to scale out to a large number of instances, we have extended MonetDB's distributed query processing feature to support predicate-based data partitioning and distributed updates. In Section 3.2, we show an example of the new features.

Appendix I includes references to the software repositories and documentation.

## 1.1 Relevance to ACTiCLOUD's Objectives, Use Cases and Business Scenarios

Table 1 shows an overview of the MonetDB artefacts we have implemented in the context of T3.3 and their related ACTiCLOUD objectives, use cases and business scenarios.

The estimation given by MALCOM helps one to choose a suitable set of resources for a certain query workload. Hence, MALCOM directly contributes to the strategic objectives SO1.1 "resource efficiency" and SO1.2 "performance stability". Businesswise, MALCOM can help us make more efficient use of the available resources, hence, it is related to business scenarios BS1 "effective consolidation for increased revenue and reduced TCO" and BS3 "Hosting larger workloads".

MDBconductor directs queries to VMs with most suitable resources and automatically resize VMs to match the resource needs of query workloads when needed. Hence, MDBconductor contributes to both strategic objectives and their sub-objectives, and the business scenarios BS1 "effective consolidation for increased revenue and reduced TCO" and BS3 "Hosting larger workloads".

MonetDBLite-Java makes it easier for application developers to deploy and manage (resources allocated to) their MonetDB databases. Hence, MonetDBLite-Java contributes to the strategic objectives SO1.1 "resource efficiency" and SO2.1 "Scalability in resource provisioning", and the business scenario BS1 "effective consolidation for increased revenue and reduced TCO".

Distributed query processing increases MonetDB's ability to handle larger workloads by scaling out. Hence, it contributes to the strategic objectives SO1.2 "performance stability" and SO2.1 "Scalability in resource provisioning", and the business scenario BS3 "Hosting larger workloads".

All four MonetDB artefacts directly or indirectly contributed to the support of analytical relational database applications with various types of workload, i.e. intermittent versus continuous uptime



and predictable versus unpredictable workload burst. Therefore, they are all related to the use cases UC2, UC3, UC4, and UC5.

**Table 1: List of T3.3 software deliverables and relation with ACTiCLOUD’s strategic objectives, use cases and business scenarios.**

No	Software Artefact	Description	Related SOs <sup>2</sup>	Related UCs <sup>3</sup>	Related BSs <sup>4</sup>
1	MALCOM	MonetDB memory footprint estimator	1.1, 1.2	2, 3, 4, 5	1, 3
2	MonetDBLite-Java	Embedded MonetDB for Java	1.1, 2.1	2, 3, 4, 5	1
3	Distributed query processing	Transparent distributed query processing on (local) MonetDB clusters	1.2, 2.1	2, 3, 4, 5	3
4	MDBconductor	Manages MonetDB VMs in the cloud, directs incoming queries to suitable VMs, and resizes VMs for better resource provisioning.	1.1, 1.2, 2.1, 2.2	2, 3, 4, 5	1, 3

<sup>2</sup> The strategic objectives are: SO1: Effective utilisation of cloud resources (SO1.1: Resource efficiency; SO1.2: Performance stability) and SO2: Deployment of resource demanding applications in the cloud (SO2.1: Scalability in resource provisioning; SO2.2: Elasticity in resource provisioning)

<sup>3</sup> The involved use cases are: UC2. Database applications with constant workload and intermittent uptime; UC3. Database applications with constant workload and continuous uptime; UC4. Database applications with predictable workload burst; and UC5. Database applications with unpredictable workload burst.

<sup>4</sup> The involved business scenarios are: BS1. Effective consolidation for increased revenue and reduced TCO, and BS3. Hosting larger workloads.

## 2 MonetDB in the (ACTi)CLOUD

There are two main challenges in running a database in cloud environments: i) to determine the compute resources needed for the duration of a session and the execution of complex business analytics queries, and ii) to exploit the locality of the persistent data in either local attached storages or remote global file servers. This leads to a plethora of design decisions related to the functionality offered by the underlying cloud infrastructure and the ability to pass provisioning information from the DBMS to this infrastructure.

In the second period of the ACTiCLOUD project, our work has continued focusing on the proper provisioning of memory resources, because the performance of MonetDB is most quickly and significantly affected by the available memory. Henceforth, with “resources” we refer to the available memory on a MonetDB virtual machine, unless otherwise explicitly stated.

In this section, we describe the MonetDB extensions we have introduced in the second half of the ACTiCLOUD project to make MonetDB more flexible in resource provisioning in cloud environments in general, and in ACTiCLOUD-enabled environments in particular. We first give a recap of MALCOM, the MonetDB memory footprint estimator reported in D3.3, before presenting MDBconductor, a new component to conduct MonetDB VM instances in cloud environments with flexible resource provisioning.

### 2.1 MALCOM recap

Scale-out of big data analytics applications often does not pay off due to the poor performance in response time and the increasing costs due to a longer execution time on a resource limited machine. To enable a stable DBMS workload environment it helps to maintain several virtual machines with different resource configurations (CPU, memory, disk, etc) hosting part of the database<sup>5</sup>, so that users can send their tasks to those machines that have the best price/performance characteristics. This, however, requires a method to decide which VM should be used for a given query.

When choosing the VM, the memory usage of a query is a particularly important factor, especially for the main-memory (optimised) DBMSs which are generally used for analytical queries today. Therefore, in the first half of the ACTiCLOUD project, we have developed MALCOM (reported in deliverable D3.3 “MonetDB ACTiCLOUD extensions v1.0”), a MonetDB memory footprint predictor for queries based on resilient intermediates in MonetDB. Unlike traditional cost-based approaches, MALCOM uses an empirical approach (i.e. using the memory usage information of queries executed in the past) to incrementally update its model to improve its predictions. The workflow of MALCOM is shown in Figure 2:

- *Step 1:* before executing a query, one can ask MALCOM to give an estimation of how much memory will be needed for this query.
- *Step 2:* MALCOM computes and returns an estimated memory footprint based on the actual memory footprint of previously executed queries kept in its dictionary.

---

<sup>5</sup> Here we are thinking about sharded databases that can have distinct partitions, replicated partitions and even partially overlapping partitions.

- *Step 3*: the user executes the query on a MonetDB server, possibly making use of MALCOM's output<sup>6</sup>.
- *Step 4*: the MonetDB server returns query results and piggy-backs the query execution traces which contain, among others, the actual memory footprint.
- *Step 5*: the user can pass the query traces to MALCOM to refine or extend its dictionary information.

A prototype implementation of MALCOM was demonstrated during the P1 review. After that, it has been published in the XtremeCLOUD workshop in December 2018<sup>7</sup>, in which, among others, experiment results of both the industrial standard benchmark TPC-H and the real-world based statistical benchmark Air Traffic were presented.

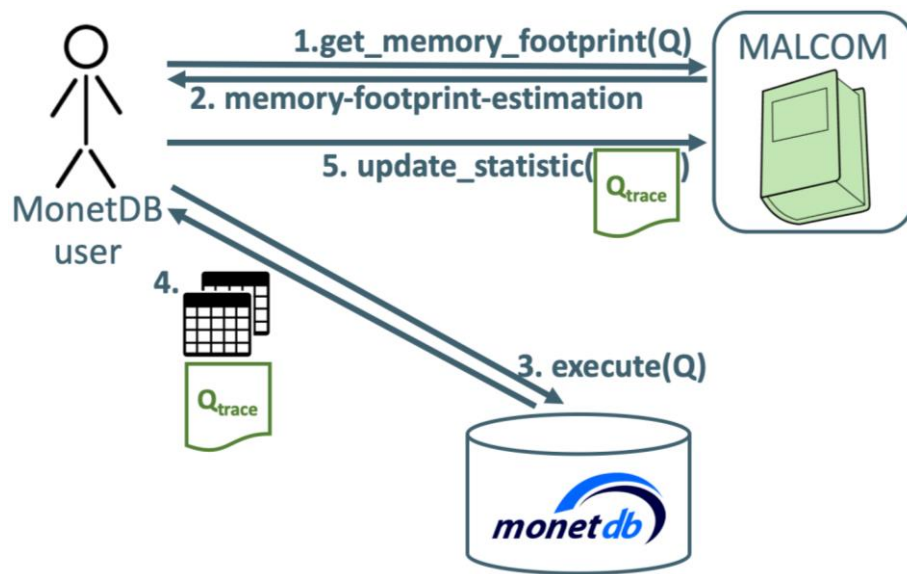


Figure 2: MALCOM workflow

## 2.2 Database server conductor in the cloud

Equipped with MALCOM's estimation, our next challenge is to find the right database backend to support a given application workload. This implies scale-up or scale-down in the case that a single DBMS instance is used, or to forward the workload to the instance with the proper resources if there are multiple DBMS instances. Therefore, in the second half of ACTiCLOUD, we have developed

<sup>6</sup> For example, if there are multiple database instances from which one can choose for the execution of a query, one can use the MALCOM estimation to make the choice; if there is only one database instance, one can use the MALCOM estimation to further estimate the query execution time, e.g. if the estimated memory footprint is much bigger than the available memory, this query is most probably going to take a long time.

<sup>7</sup> M. Kersten, Y. Zhang, P. Katsogridakis, P. Koutsourakis, and J. van Ruth. "Database Resource Allocation Based on Resilient Intermediates," in *Proceedings of 2018 IEEE International Conference on Cloud Computing Technology and Science*, CloudCom 2018, Nicosia, Cyprus, December 10-13, 2018.

*MDBconductor*, an auxiliary tool that manages a cluster of MonetDB instances in the cloud and decides “when” and “what” to do for each instance.

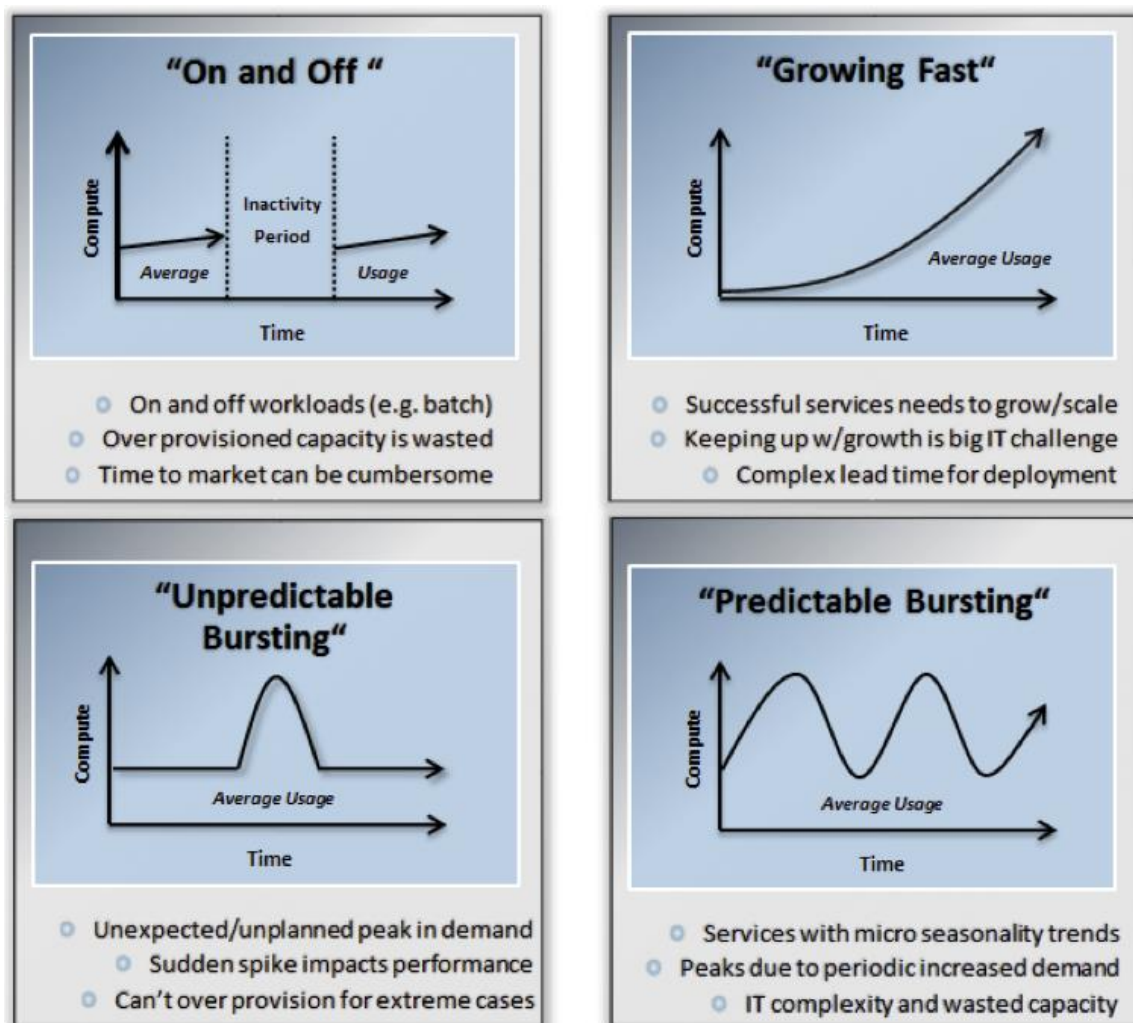


Figure 3: Common workload patterns of business analytics applications.

The boundary conditions within ACTiCLOUD are to focus on business analytics applications, which typically run a large collection of read-only queries over a long period of time. Their workloads can be classified as one or more of the patterns shown in Figure 3. To handle those workloads efficiently in terms of query processing time and/or costs, one can either prepare multiple database instances with different sizes, or resize an existing database instance on-demand (or both). *MDBconductor* supports both cases, but in this deliverable, we focus on resizing a database instance, because it is a bigger challenge.

Furthermore, we focus on memory resources, because they determine to a large extent the cost of provisioning an instance and the performance of MonetDB. Cutting down the memory requirements by a factor 2, often reduces the cost by a factor of 2<sup>8</sup>. Of course, the time it takes to provision a larger machine can take several minutes. How much of this process can be sped up

<sup>8</sup> <https://aws.amazon.com/ec2/pricing/on-demand/>

depends on both the facilities provided by the underlying cloud platforms and their costs. For instance, on AWS, it is possible to prepare pre-warmed instances that can be brought up-and-running rapidly<sup>9</sup>.

### The architecture of MDBconductor

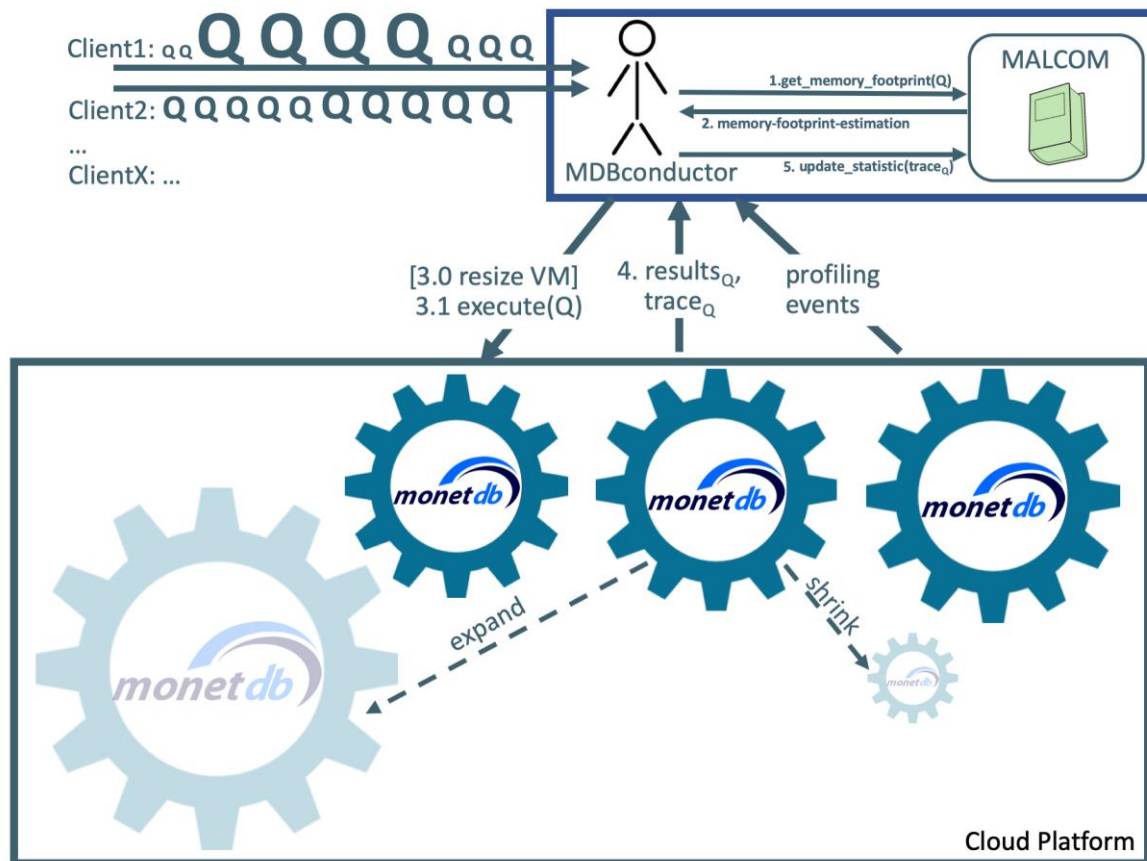


Figure 4: MDBconductor architecture and workflow

The architecture of MDBconductor is shown in Figure 4, in which different sizes of “Q” indicate queries with different data and/or computation intensities, and gears in different sizes in the “Cloud Platform” box represent MonetDB VM instances with different resources.

The main tasks of MDBconductor are: i) managing a cluster of MonetDB VM instances on a cloud platform, e.g. AWS and OpenStack; ii) listening to incoming client queries, and using the memory footprint estimations given by MALCOM to choose the most suitable MonetDB VM instances to execute a query; and iii) resizing a MonetDB VM when necessary. The MDBconductor itself can be run on premises, or in a VM or Docker container in the cloud. We run MALCOM as an auxiliary tool of MDBconductor on the same machine. More precisely, the workflow of MDBconductor is as follows (the steps are numbered in Figure 4):

<sup>9</sup><https://aws.amazon.com/about-aws/whats-new/2018/11/amazon-ec2-now-lets-you-pause-and-resume-your-workloads/>



- When MDBconductor receives a query  $Q$ , it asks MALCOM for a memory footprint estimation  $est_Q$  (i.e. steps 1 and 2).
- Then MDBconductor selects or prepares a MonetDB VM instance to execute this query according to the following (simplified) logic:
  - If a MonetDB VM instance with sufficient resources (i.e. the available memory is within the range  $[est_Q, est_Q * 2.5]$ ) already exists, this instance can be directly used to execute the query (i.e. step 3.1);
  - Otherwise, MDBconductor will shrink or expand an existing instance to the smallest instance with a memory size  $\geq est_Q * 1.3$  and migrate the database<sup>10</sup> before using it to execute the query (i.e. conduct the optional step 3.0 before 3.1).
- Finally, query results and traces are produced by the MonetDB instance and sent to the MDBconductor, which in turn takes care of returning the query results to the user and updating MALCOM with the newly received information (i.e. steps 4 and 5).

The ranges above were chosen because in the current set-up we prefer shorter query execution time over lower cloud costs. Since MonetDB is memory hungry, even a small shortage of memory can have a large negative impact on the query performance. Therefore, we choose to expand a small VM much quicker than shrink a large one.

Of course, in real applications, one would not immediately resize a VM for every single query with a peak in its resource requirement. Therefore, it is configurable how sensitive MDBconductor should be in changing resource requirements of the queries. Hence, in the case of all four workload patterns in Figure 3, sometime after the workload has changed, MDBconductor will automatically decide to expand or shrink the VMs in its cluster.

Currently, MDBconductor and the MonetDB VM instances run on AWS. MDBconductor is an external tool for MonetDB implemented using Python, Ansible and Terraform. A next step is to port MDBconductor to the OpenStack-based cloud platform provided by OnAPP.

### Database Performance indicators

As shown in Figure 4, MDBconductor also continuously collects profiling events produced by the MonetDB servers so as to monitor their states. To monitor the resource consumption specific to individual queries, we have extended the MonetDB kernel with a new performance indicator, which has been added to the profiling events.

MonetDB has been primarily designed for multi-core parallel execution. Its column-at-a-time query execution model is similar to the SIMD technique<sup>11</sup>: each relational operator is applied on a whole column<sup>12</sup>. The column is divided into the same number of chunks as the number of available CPU cores and the same operator is applied on all chunks in parallel. Given sufficient resources, MonetDB runs at full speed at the maximal level of parallelism. However, with insufficient resources, this can cause resource contention and/or attracts the attention of the OOM-killer<sup>13</sup>. To alleviate these downsides, MonetDB has a built-in self-protection mechanism: if at runtime, it detects that there is insufficient memory resources, MonetDB will automatically reduce the level

<sup>10</sup> The precise migration strategy depends on the supported features of the underlying cloud platform. On AWS, this is currently done by detach the EBS storage from the old VM and attach it to the new VM.

<sup>11</sup> Single Instruction Multiple Data (SIMD), <https://en.wikipedia.org/wiki/SIMD>

<sup>12</sup> Or all values of a column that have been selected by a select operator.

<sup>13</sup> <https://www.kernel.org/doc/gorman/html/understand/understand016.html>

of parallelism. It postpones the execution of new relational operators until sufficient memory resources have become available<sup>14</sup> after some of the current relational operators have finished their executions.

Holding back due to resource limitations is a peculiar performance indicator. On the one hand, it is a strong signal that a query should be moved to a larger (virtual) machine, but on the other hand, it cannot be detected by any external monitoring tools, because when a system's resources are fully utilised, there is no way for those tools to tell if the resources are sufficient. This information has to come directly from the execution engine of the database itself. Therefore, we have extended the MonetDB kernel to keep track of the number of query execution threads<sup>15</sup> that are kept waiting and regularly reveal this information as heartbeat profiling events, by default every 50ms.

Table 2: a JSON object containing one heartbeat event

```
{ "source": "heartbeat",
  "session": "5b3d1a1d-cc37-451e-893c-d0a92ea38e02",
  "clk": 22999067,
  "ctime": 1565785697263747,
  "rss": 89,
  "nvcsw": 8,
  "state": "ping",
  "cpuload": [0.75, 0.6, 0.5, 0.67],
  "waiting": 0 }
```

Table 2 shows an example JSON object containing one heartbeat event:

- a MonetDB server outputs various profiling events, with “source” we can filter out the “heartbeat” events;
- “session” is a UUID per server session;
- “clk” is the number of milliseconds since the start of this MonetDB server session;
- “ctime” is the number of milliseconds since the UNIX epoch;
- “rss” is the resident set size of this server process in MB;
- “nvcsw” is the number of non-voluntary context switch since the start of this server session;
- “state” is mainly used to denote the execution state of a relational operator, which is either “start” or “end”, “ping” is a special state for the heartbeats;
- “cpuload” is the CPU load per core; and finally
- “waiting” is the number of worker threads waiting.

MDBconductor continuously collects profiling events produced by the MonetDB servers, specifically the heartbeat events and query execution times. Thus, the estimation of MALCOM helps us in advance to avoid performance degradation to some extent, while the heartbeats of the number of waiting threads can tell us if a query has actually suffered from any performance

<sup>14</sup> The threshold is ~80%, i.e. when starting a new relational operator, the total memory consumption of the system should not be larger than 80% of the total available memory on this machine.

<sup>15</sup> Each thread executes one relational operator at a time.

degradation. MDBconductor stores the profiling information in its logs for future inspection or to convert them into performance indicators that can be communicated to external tools or users.

**MonetDB and ACTiManager.** The profiling information collected by MDBconductor creates an opportunity for the eventual integration with ACTiManager, which is an ongoing work under the context of WP4 (“Integration, validation and evaluation”).

ACTiManager allows applications to incorporate special functionality to expose their desired metrics of interest and achieved Quality of Service (QoS). When this functionality is provided by the application, ACTiManager will be able to utilise it to act towards maintaining the desired QoS of applications<sup>16</sup>. For this purpose, ACTiManager provides a PerformanceAgent, which implements a REST interface to capture performance alerts from the applications.

Based on the profiling events produced by the MonetDB database servers under its responsibility, MDBconductor can compute if a database server is suffering from performance degradation or not. The MDBconductor can then turn on and off a performance alert by sending a REST PUT request to the ACTiManager PerformanceAgent. With a GET request, the MDBconductor can check the current status at any time.

The challenge here is deciding when a performance alert should be sent, because it is affected by at least three non-trivial factors: i) the (recent) behaviour of the DBMS within a user session, ii) the prediction of the future workload, and iii) the time required to take actions (e.g. migrate or resize a VM). Predicting future workload is only feasible for stable workloads, hence, it would not work in interactive sessions without human intervention. In the current implementation we use the policy “*the recent workload is a good predictor for the short-term future*”. Concerning the time required to take actions, ACTiManager may provide an estimation of how long it would take to mitigate interference or provision a new instance. A VM migration may involve migration of the database itself as well, which might be realised by (re\_)attaching the associated SSD, or by using a secondary copy of the database volume, or a disk migration.

The results of this integration will be reported in D4.5 “ACTiCLOUD final evaluation” due M36.

---

<sup>16</sup> G. Goumas, V. Karakostas, K. Nikas, D. Siakavaras, S. Psomadakis, S. Gerangelos, E. B. Lakew, P. Svärd, and S. Kollberg. “D2.2: Distributed Cloud Resource Manager v1.0,” ACTiCLOUD deliverable, July, 2018.



### 3 Updates of D3.3

In the previous deliverable D3.3 “MonetDB ACTiCLOUD extensions v1.0”, we have reported our work on MonetDBLite-Java and distributed query processing in MonetDB. In the second period of this project, we have continued developing and improving these MonetDB features. In this section, we give an update of the work we have done in the second period on them.

#### 3.1 MonetDBLite-Java

In D3.3, we have presented the first stable version of MonetDBLite-Java, an embedded version of MonetDB. We have described its connection APIs, distribution and evaluation against two of the most popular embedded databases for Java, i.e. SQLite and H2. In D4.3, we have integrated MonetDBLite-Java with HyperscaleJVM, the JVM provided by the ACTiCLOUD partner UNIMAN, and evaluated the integration against HotSpot JVM, which is the JVM for Oracle JDK.

In the second period of this project, the work we have done around MonetDBLite-Java focuses on broadening its usability, i.e. ported MonetDBLite-Java to ARM64 architecture, and improving its long term maintainability, i.e. merge the MonetDB and MonetDBLite code base. Below we shortly describe both works.

##### MonetDBLite-Java on ARM64

MonetDBLite-Java deploys the native MonetDBLite library in a JVM through the standard JDBC API or its own Embedded API. This means that unlike regular Java programs, MonetDBLite-Java is both operating system and architecture dependent. The previously released MonetDBLite-Java library was only available to the x86\_64 architecture. Since one of the main hardware architectures used in the ACTiCLOUD project is ARM, we have ported MonetDBLite-Java to run in Linux on the ARM64 (AArch64) architecture. The new version can be obtained from the Maven Snapshot Repository<sup>17</sup>. To use it:

- In Apache Maven<sup>18</sup>:

```
<dependency>
  <groupId>monetdb</groupId>
  <artifactId>monetdb-java-lite</artifactId>
  <version>2.39-SNAPSHOT</version>
</dependency>
```

- In Gradle<sup>19</sup>: `implementation 'monetdb:monetdb-java-lite:2.39-SNAPSHOT'`

<sup>17</sup> At the time of this deliverable, it's not available at Maven's main repository yet.

<sup>18</sup> Note that the snapshot repositories should be allowed in the settings. See this thread for more information: <https://stackoverflow.com/questions/7715321/how-to-download-snapshot-version-from-maven-snapshot-repository#answers-header>

<sup>19</sup> Note the use of “implementation” instead of the previously used “compile”, because the former also adds runtime dependencies.

- Otherwise download the two JAR files `monetdb-java-lite-2.39-SNAPSHOT` and `monetdb-jdbc-new-2.37-SNAPSHOT` from our website<sup>20</sup> and add them to the `CLASSPATH`.

### Merge MonetDB and MonetDBLite

At the time of this writing, MonetDBLite supports five programming languages, i.e. R, Python, Java and C/C++, as shown in Figure 5. During development up to now, the codebase of MonetDBLite has diverged from MonetDB into a brand new one. Consequently, changes in both codebases had to be manually examined one-by-one and synchronised if necessary. Similarly, new features in one codebase had to be manually ported to the other, which sometimes doubled the implementation effort. Moreover, due to the separate codebases, MonetDBLite is currently not part of the MonetDB nightly testing suite. To ease the development and maintenance effort of both codebases at long term, the MonetDB team decided to merge the two codebases into one, before adding new features to MonetDBLite. However, as a result of this merge, new features of MonetDB are now automatically added to MonetDBLite. On the other hand, features used to be available solely for MonetDBLite, e.g the in-memory (only) mode which is of particular interest for ACTiCLOUD, have been backported to MonetDB. So, once the merge is finished, there will be one codebase for both MonetDB and MonetDBLite: every new feature and bugfix will be immediately available to both databases. At the same time, a single build, installation and testing system is available as well. Below we highlight several of the main changes.

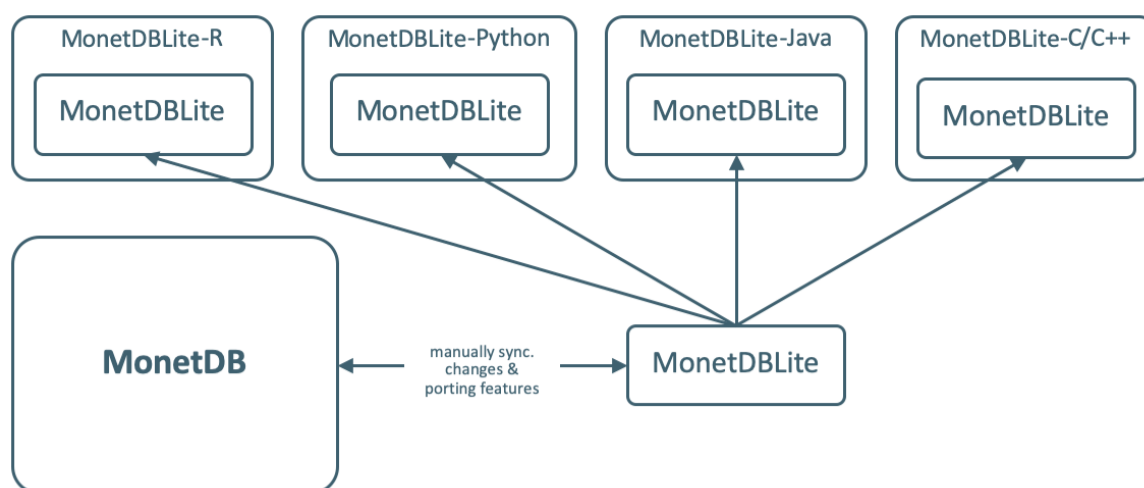


Figure 5: MonetDB versus MonetDBLite codebases

First, the in-memory mode was featured in MonetDBLite, with which a MonetDB database runs exclusively inside the system's main memory. This implies no data will be persisted into non-volatile memory. In-memory databases provide reduced latencies derived from no disk access compared to persisted databases. This feature is beneficial for caching and executing on-the-fly data analytics. During our codebase merging process, this feature has been ported to the current main development branch of MonetDB, and it will be available in the next MonetDB feature release planned in Q4 2019. Then, one can instruct a MonetDB server to operate only in memory by starting the server using the option `--in-memory`.

<sup>20</sup> <https://www.monetdb.org/downloads/Java-Experimental>

Second, for ease of deployment, MonetDBLite has embedded SQL and MAL<sup>21</sup> start-up scripts into its binaries. This implied diverged start-up methods between the two MonetDB versions. During the codebase merge, this feature has been backported into MonetDB. With embedded start-up scripts, a MonetDB server starts faster because it no longer spends I/O time to access start-up scripts on disk. Further, to ease the back-porting of MonetDBLite features to MonetDB, we have developed auxiliary tools, to automate as much porting-work as possible.

Finally, we have also decided to attempt to migrate MonetDB's existing building system into a more modern and industry standard one. So far, MonetDB has always been built using GNU Autotools<sup>22</sup> for UNIX platforms, generated Nmake<sup>23</sup> files on Windows and a single Makefile for MonetDBLite. With different build scenarios for each setup, it makes code building difficult to maintain; in addition, Autotools hasn't been under active development for several years. Therefore the team searched for a viable alternative to it.

CMake<sup>24</sup> is a cross-platform software tool used to manage software deployment across heterogeneous platforms, with wide usage in the software industry. During the merging of MonetDB and MonetDBLite codebases, the team has also started to migrate the build scripts of MonetDB to CMake. This migration allows to unify all building setups, resulting in less boilerplate building code and easier maintenance. In addition, some experiments show that the compilation time of CMake-generated Makefiles is about 15% faster compared to the Autotools generated ones. Another benefit of CMake is the generation of configuration files to popular IDEs in the market such as Visual Studio on Windows and XCode on MacOS. The major advantage of these setups is the ability to use the debugger on the IDE instead of the console. Currently, a first implementation of using CMake to build and install MonetDB is available as rpm files for Red Hat based Linux distributions, deb files for Debian based Linux distributions, a Windows installer and an Apple Disk Image (i.e. dmg file) for MacOS. A first official release is planned for 2020/H1, after which we will gradually move the whole MonetDB compilation, installation and testing system to use CMake.

## 3.2 Distributed Query Processing

### Basic concepts

To allow querying data distributed over multiple MonetDB servers, we have introduced three new types of SQL tables: `REMOTE TABLE` (with which one can create a virtual table on one MonetDB server to refer to a table physically located on another MonetDB server), `REPLICA TABLE` (with which one can create a virtual replica table to contain tables located on the local server or a remote server. All tables in a replica table are exact replicas of each other) and `MERGE TABLE` (with which one can create a virtual merge table to contain tables located on the local server or a remote server. All tables in a merge table have the same signature and contain possibly overlapping partitions of a big table).

---

<sup>21</sup> MAL (MonetDB Assembly Language) is MonetDB's low level language used for queries execution

<sup>22</sup> [https://www.gnu.org/software/automake/manual/html\\_node/Autotools-Introduction.html](https://www.gnu.org/software/automake/manual/html_node/Autotools-Introduction.html)

<sup>23</sup> <https://docs.microsoft.com/en-us/cpp/build/reference/nmake-reference?view=vs-2019>

<sup>24</sup> <https://cmake.org>

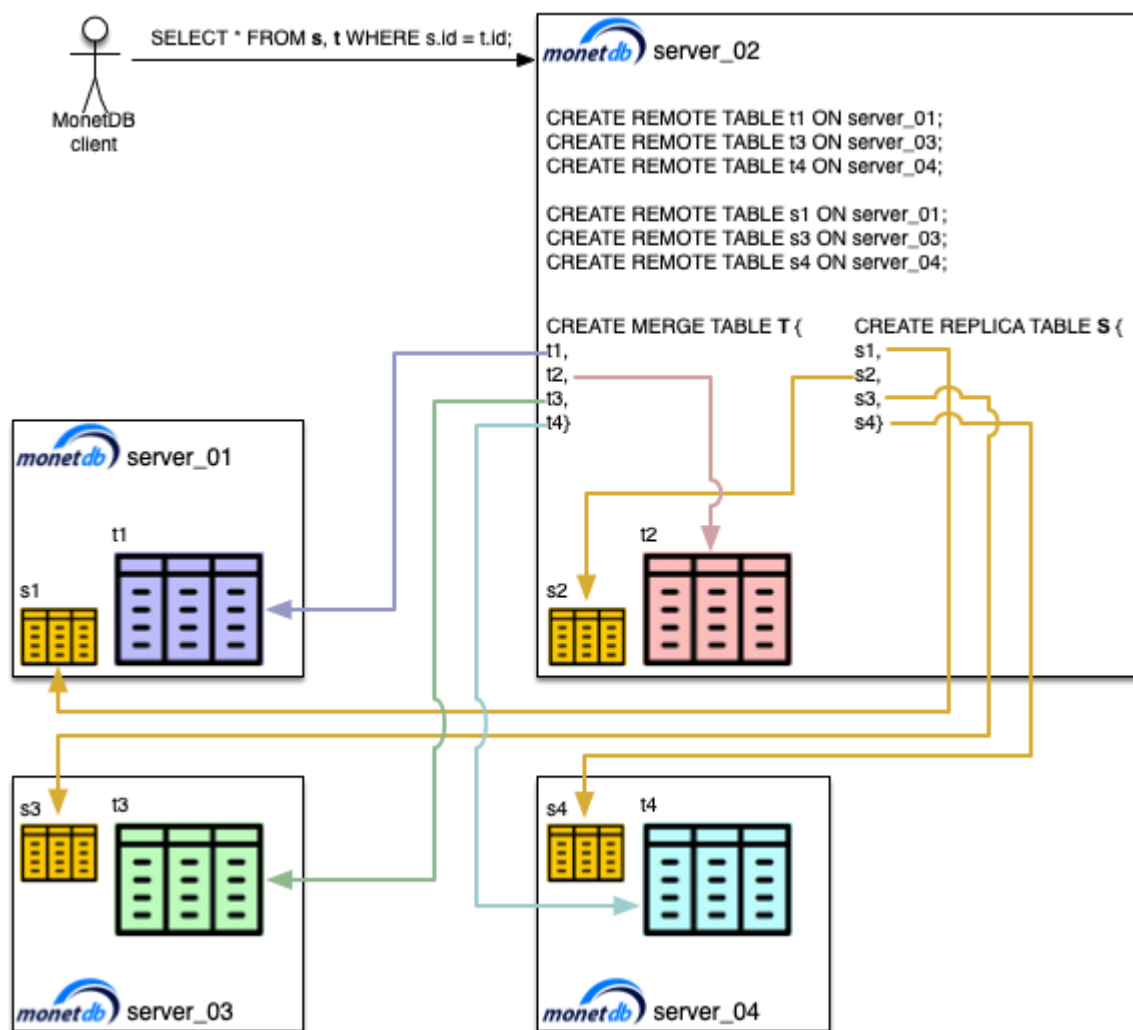


Figure 6: query distributed data in a MonetDB cluster: tables `s1`, `s2`, `s3`, `s4` are exact copies of each other (shown in the same colour). This fact can be indicated using `CREATE REPLICAS TABLE`, so that when the table `S` is queried, the query will be run using the nearest replica. Tables `t1`, `t2`, `t3`, `t4` contain data partitions of a bigger table (shown in different colours). One can merge a number of partitions using `CREATE MERGE TABLE`, so that when the table `T` is queried, the query will be run on all partition tables included in `T`.

Figure 6 depicts a cluster containing four MonetDB servers with several tables spread amongst them: the tables `s1` - `s4` contain the same data replicated across all servers, while the tables `t1` - `t4` are partitions of a big table and distributed over the servers. Each MonetDB server has the same query capability, so to query all data in this cluster, we can choose any one of these servers, e.g. *server\_02*, as the master instance. On *server\_02*, we first register `t1`, `t3`, `t4` and `s1`, `s3`, `s4` as `REMOTE TABLE`-s. Then we add `s1` - `s4` to a `REPLICAS TABLE S`, and `t1` - `t4` to a `REMOTE TABLE T`. Finally, our “MonetDB client” can query the tables `S` and `T` on *server\_02* as if they are normal SQL tables. In addition, a MonetDB client can now query the table `s1`, `s3`, `s4` and `t1`, `t3`, `t4` both through *server\_02* and directly on the corresponding *server\_01*, *server\_03* and *server\_04*.

MonetDB’s support to run distributed read-only queries has been described in D3.3, in which we have also evaluated this feature using the Air Traffic benchmark on KMAX. However, this version does not allow any updates through `MERGE TABLE`-s and `REMOTE TABLE`-s. Users are expected to

prepare the whole setup themselves: replicate or partition the data, distribute it over the MonetDB servers and load the data on each MonetDB server into the designated local tables.

### Automated data partitioning and distribution

If we truly want to scale out MonetDB to many instances, we must automate the process of data partitioning and distribution. Therefore, in the second period of the ACTiCLOUD project, we have started to extend `MERGE TABLE` and `REMOTE TABLE` to support predicate-based data partitioning and distributed updates.

First, we have extended the definition of `MERGE TABLE` with a `PARTITION BY` clause, so that one can specify how the data should be partitioned (by range or value) among the child tables of a `MERGE TABLE`. Then, we have extended `MERGE TABLE` to allow `INSERT`, `UPDATE` and `DELETE`. Finally, we have extended `REMOTE TABLE` to allow `INSERT`. Below we show a small example of this new feature in action.

First, we set up a MonetDB cluster with a master instance (i.e. 'mst') and two worker instances (i.e. 'wk1' and 'wk2'):

```
$ monetdbd create acticloud
$ monetdbd start acticloud
$ monetdb create mst; monetdb create wk1; monetdb create wk2
created database in maintenance mode: mst
created database in maintenance mode: wk1
created database in maintenance mode: wk2
$ monetdb start mst; monetdb start wk1; monetdb start wk2
starting database 'mst'... done
starting database 'wk1'... done
starting database 'wk2'... done
$ monetdb status
name  state  health  remarks
wk2   LR 5s  100%  0s
wk1   LR 5s  100%  0s
mst   LR 6s  100%  0s
```

Next, we create a table in each of these databases to hold some time series data (i.e. each record contains a timestamp and a value):

```
$ mclient -d wk1 -s 'CREATE TABLE first_decade(ts TIMESTAMP, val INT);'
operation successful
$ mclient -d wk2 -s 'CREATE TABLE second_decade(ts TIMESTAMP, val INT);'
operation successful
$ mclient -d mst -s 'CREATE TABLE later_decades(ts TIMESTAMP, val INT);'
operation successful
```

Then, we register the two tables hosted on the workers at the master as `REMOTE TABLE`-s:

```
$ mclient -d mst -s "CREATE REMOTE TABLE first_decade(ts TIMESTAMP, val INT) ON
'mapi:monetdb://localhost:50000/wk1';"
operation successful
```

```
$ mclient -d mst -s "CREATE REMOTE TABLE second_decade(ts TIMESTAMP, val INT) ON
'mapi:monetdb://localhost:50000/wk2';"
operation successful
```

Finally, on the master database, we create a `MERGE TABLE` *timeseries* into which we will merge all tuples stored in *first\_decade*, *second\_decade* and *later\_decades*. The data in *timeseries* will be partitioned into different ranges by their *ts* values.

```
$ mclient -d mst -H
Welcome to mclient, the MonetDB/SQL interactive terminal (unreleased)
Database: MonetDB v11.33.8 (hg id: 75e955be90d0),
'mapi:monetdb://Nyx.local:50000/mst'
Type \q to quit, \? for a list of available commands
auto commit mode: on
sql>CREATE MERGE TABLE timeseries (ts TIMESTAMP, val INT) PARTITION BY RANGE ON
(ts);
operation successful
```

Next, we add the child tables into the merge table. This is also the place where we should specify which value range a child table contains:

```
sql>ALTER TABLE timeseries ADD TABLE first_decade AS PARTITION FROM TIMESTAMP
'2000-01-01 00:00:00' TO TIMESTAMP '2010-01-01 00:00:00';
operation successful
sql>ALTER TABLE timeseries ADD TABLE second_decade AS PARTITION FROM TIMESTAMP
'2010-01-01 00:00:00' TO TIMESTAMP '2020-01-01 00:00:00';
operation successful
sql>ALTER TABLE timeseries ADD TABLE later_decades AS PARTITION FROM TIMESTAMP
'2020-01-01 00:00:00' TO RANGE MAXVALUE WITH NULL VALUES;
operation successful
```

From now on, we can rely on the merge table for data partitioning and distribution:

```
sql>INSERT INTO timeseries VALUES
more>(TIMESTAMP '2000-01-01 00:00:00', 1),
more>(TIMESTAMP '2002-12-03 20:00:00', 2),
more>(TIMESTAMP '2012-05-12 21:01:00', 3),
more>(TIMESTAMP '2019-12-12 23:59:59', 4),
more>(TIMESTAMP '2005-02-13 01:00:00', 2000),
more>(TIMESTAMP '2020-01-01 00:00:00', 5),
more>(NULL, 6);
7 affected rows
```

We check that the records have been inserted into the correct tables:

```
sql>SELECT * FROM timeseries;
+-----+-----+
| ts                | val |
+=====+=====+
| 2000-01-01 00:00:00.000000 | 1 |
```

```

| 2002-12-03 20:00:00.000000 | 2 |
| 2005-02-13 01:00:00.000000 | 2000 |
| 2012-05-12 21:01:00.000000 | 3 |
| 2019-12-12 23:59:59.000000 | 4 |
| 2020-01-01 00:00:00.000000 | 5 |
| null | 6 |
+-----+-----+
7 tuples
sql>SELECT * FROM first_decade;
+-----+-----+
| ts | val |
+-----+-----+
| 2000-01-01 00:00:00.000000 | 1 |
| 2002-12-03 20:00:00.000000 | 2 |
| 2005-02-13 01:00:00.000000 | 2000 |
+-----+-----+
3 tuples
sql>SELECT * FROM second_decade;
+-----+-----+
| ts | val |
+-----+-----+
| 2012-05-12 21:01:00.000000 | 3 |
| 2019-12-12 23:59:59.000000 | 4 |
+-----+-----+
2 tuples
sql>SELECT * FROM later_decades;
+-----+-----+
| ts | val |
+-----+-----+
| 2020-01-01 00:00:00.000000 | 5 |
| null | 6 |
+-----+-----+
2 tuples

```

The definition of the new syntax and more examples can be found on MonetDB website<sup>25</sup>. Updatable merge tables with predicate-based data partitioning have already been released in the Apr2019 version of MonetDB<sup>26</sup>. Updatable remote tables are under active development; an official release (which should support not only INSERTs, but also UPDATES and DELETES, as well as ACID properties for distributed transactions) is in our long term roadmap.

<sup>25</sup> For instance, <https://www.monetdb.org/Documentation/Manuals/SQLreference/SQLSyntaxOverview>, and <https://www.monetdb.org/blog/updatable-merge-tables>.

<sup>26</sup> <https://www.monetdb.org/Downloads/ReleaseNotes>

## 4 Conclusion

In this deliverable, we have first presented MDBconductor, a new MonetDB artefact that manages flexible resource provisioning of MonetDB VMs and query executions in the cloud; then we have given updates to MonetDBLite-Java and distributed query processing features of MonetDB.

A prototype of MDBconductor has been implemented, which will be integrated with the ACTiManager and evaluated in the context of WP4. After ACTiCLOUD has ended, we will continue developing MDBconductor as a component of the future MonetDB DBaaS offering.

MonetDBLite-Java and distributed query processing are already part of the official MonetDB open-source software suite. Parts of our work in the second period have already been released. The remaining parts are planned for releases in 2020 and beyond.



## Appendix I References

### Software repositories

- MALCOM: <https://github.com/MonetDBSolutions/malcom/>
- MonetDBLite-Java: <https://www.monetdb.org/downloads/Java-Experimental>
- Distributed updatables: <https://dev.monetdb.org/hg/MonetDB/shortlog/acticloud>
- MDBconductor is currently under development and maintained in a private repository in <https://github.com/MonetDBSolutions>. MDBconductor will be open-sourced before the end of the project, and an updated URL to the repository will be provided in deliverable D4.5 (due M36).

### Software documentations

- MALCOM: <https://github.com/MonetDBSolutions/malcom-docs/>
- MonetDBLite-Java: <https://www.monetdb.org/blog/monetdblite-for-java>
- Updatable merge tables: <https://www.monetdb.org/blog/updatable-merge-tables>