Co-funded by the European Commission within the Seventh Framework Programme

Project no. 318521

HARNESS

Specific Targeted Research Project
HARDWARE- AND NETWORK-ENHANCED SOFTWARE SYSTEMS FOR CLOUD COMPUTING

Industrial Requirements Report

D2.2

Due date: 1 July 2013
Submission date: 15 July 2013

Start date of project: 1 October 2012

Document type: Deliverable
Activity: RTD
Work package: WP2

Editor: Oliver Pell (MAX)

Contributing partners: MAX, SAP, IMP

Reviewers: WP Leaders

Dissemination Level

PU	Public	√
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Revision history:

Version	Date	Authors	Institution	Description
0.1	2013/05/01	Oliver Pell	MAX	Outline
0.2	2013/06/19	Oliver Pell, John McGlone, Alexandros Koliousis, Gabriel Figueiredo	MAX, SAP, IMP	Initial content for applications
0.3	2013/06/24	Peter Sanders	MAX	Added conclusions
0.4	2013/06/25	Gabriel Figueiredo	IMP	Consistency checks and edits
0.5	2013/06/26	Guillaume Pierre, Oliver Pell, John McGlone, Alexandros Koliousis	UR1, MAX, SAP, IMP	Edits
1.0	2013/07/08	Alexander Wolf	IMP	Final review and edits by Coordinator

Tasks related to this deliverable:

Task No.	Task description	Partners involved°
T2.2	Analyse industrial requirements	MAX*, SAP

°This task list may not be equivalent to the list of partners contributing as authors to the deliverable
*Task leader

Executive Summary

The *Hardware- and Network-Enhanced Software Systems for Cloud Computing* (HARNESS) project aims to advance the state-of-the-art in cloud data centres to enable cloud providers to profitably offer and manage the tenancy of specialised hardware and network technologies, while also enabling software developers to seamlessly, flexibly and cost-effectively integrate these technologies with their cloud-hosted applications.

This report describes the three industrial use cases that will be used throughout the project to validate the HARNESS platform.

HPC: Reverse Time Migration A proprietary implementation of a geoscience algorithm simulating wave propagation through the earth.

In-Memory Database: Delta Merge A proprietary database operation merging a temporary delta store into a main store.

Data-Flow Computing: AdPredictor An open-source statistical learning algorithm for Web advertisement click-through rate.

The use cases have been chosen to represent a cross section of cloud-based applications, while also attaining tangible benefit from their deployment on specialised technologies. The use cases can take advantage of specific technologies for computation, communication and storage that will give the HARNESS platform a competitive advantage over current state-of-the-art cloud platforms.

Analysis of each of these use cases has resulted in a set of requirements for each. Generally, the requirements describe a platform that is flexible in providing resources to applications. The resources being provisioned depend on customer directives, application algorithms and available suitable resources. Furthermore, the requirements describe a platform that is dynamic, using appropriate heterogeneous resources when required.

Since requirements for each use case were derived independently from each other, many of them are unique and specific to a particular application domain. But there are others that are common and shared among the use cases. Some of the requirements presented in this document include:

- the cloud tenant's ability to describe application properties and requirements;
- exploitation of heterogeneous computation, communication and storage resources that allow the execution of particular algorithms in a way that meets cloud tenant's conditions;
- support for multiple algorithms allowing the platform to make choices between different versions of an application that have implications on how computation, communication and storage resources are provisioned; and
- the flexibility to adapt to changing demands of the application and the platform itself.

The requirements reported in this deliverable, along with the general requirements presented in Deliverable D2.1 [8], focus on satisfying the goals of the HARNESS project. In meeting both sets of requirements—general and application-specific—the HARNESS project will develop the next generation cloud platform that is capable of exploiting specialised technologies in three very distinct and demanding application domains.

Contents

Acronyms

AWS *Amazon Web Services*. 11, 13

CAPEX *capital expenditure*. 10

CTR *click-through rate*. 17, 18, 22, 23

DFE *data-flow engine*. 6, 25

DRAM *dynamic random-access memory*. 7, 8, 25

FPGA *field-programmable gate array*. 6, 25, 27

GPGPU *general-purpose graphics processing unit*. 6, 25, 27

HARNESS *Hardware- and Network-Enhanced Software Systems for Cloud Computing*. i, ii, 1, 4, 7–9, 15, 16, 18, 21, 24–27, 29, 31

HPC *high-performance computing*. 3

i.i.d. *independent and identically distributed*. 20, 23

IMP *Imperial College London*. 1

MAX *Maxeler Techologies*. 1

OLTP *on-line transaction processing*. 12

OPEX *operational expenditure*. 11, 13

PCIe *peripheral component interconnect express*. 14

RAM *random-access memory*. 23

RTM *reverse time migration*. 3–8, 29

SSD *solid-state drive*. 8

1 Introduction

This report describes the industrial use-case requirements for the *Hardware- and Network-Enhanced Software Systems for Cloud Computing* (HARNESS) platform.

The HARNESS project aims to advance the state-of-the-art in cloud data centres to enable cloud providers to profitably offer and manage the tenancy of specialised hardware and network technologies, while also enabling software developers to seamlessly, flexibly and cost-effectively integrate these technologies with their cloud-hosted applications. This report focuses on analysis of the requirements for three use-case applications chosen to represent three domains that could benefit from HARNESS technology:

1. **High-Performance Computing**, using project partner *Maxeler Techologies* (MAX)'s proprietary Reverse Time Migration as a use case.

2. **In-memory Databases**, using project partner SAP's HANA Delta-Merge operation as a use case.

3. **Data-Flow Computing Map/Reduce** using project partner *Imperial College London* (IMP)'s open-source AdPredictor as a use case.

Notice that the first two use-case applications are provided by the project's industrial beneficiaries, while the third is based on publicly available documentation and open-source facilities. This report describes each of the use-case applications and identifies key requirements they place on the HARNESS platform to support their efficient use in a heterogeneous cloud.

This deliverable is structured as follows. Chapters 2–4 report each of the three use-case applications. For each use case, this report describes: (a) the application domain scope and key challenges, (b) an overview of the application, (c) how the application is currently used in an industrial setting, (d) how the application could exploit the cloud platform in general and (e) how the application could exploit heterogeneous resources in particular. The specific requirements for each application are presented at the end of each chapter. ***Notice that the numbering of requirements continues the numbering used in Deliverable D2.1*** [8]. Chapter 5 summarises the application requirements from the perspective of the HARNESS platform, discussing the uniqueness and commonalities of the requirements. Chapter 6 concludes this report.

2 HPC: Reverse Time Migration

Reverse time migration (RTM) represents a class of applications that are computationally intensive and typically process large amounts of data. Historically, these have been referred to as *high-performance computing* (HPC) applications.

This chapter describes RTM, an HPC application drawn from the domain of geoscience where it provides commercially important information to the oil industry.

2.1 General Domain Scope and Requirements

HPC addresses highly complex computational problems that are due to the large amounts of data to process and/or the complexity of the involved calculation. Such tasks are typically performed on supercomputing-class systems or clusters, or on optimised parallel architectures when suitable.

Key characteristics of HPC problems are:

- *Large scale:* an HPC system will typically consist of many compute nodes, ranging from dozens to tens of thousands.
- *Long run times:* jobs will run for hours up to months at a time.
- *High computational requirements:* in aggregate, arithmetic or other processing requirements are high. However. these can often be delivered either by a number of very powerful compute nodes or by a larger number of less powerful compute nodes.
- *High memory requirements:* many HPC problems utilise large volumes of data in memory, which can be located in each node or spread across many compute nodes. Within each node, the computations generally require very high bandwidth to memory.
- *Limited user interactivity:* due to the long-running nature of most HPC applications, even on the largest supercomputers, user interaction is normally relatively limited, for example setting up a job and then awaiting its completion.
- *Fast interconnect:* HPC systems often have faster inter-node interconnects compared to standard computer servers or desktop systems. This is required to satisfy inter-node communication during compute-job execution. At the same time, many HPC jobs may be, at least to some degree, "embarrassingly parallel" and require minimal inter-node bandwidth, with communication bandwidth instead utilised for reading/writing data to/from storage.

2.2 RTM Application Description

Some of the most computationally intensive geoscience algorithms involve simulating wave propagation through the earth. The objective is typically to create an image of the subsurface from acoustic

measurements performed at the surface. To create an image of the subsurface, a low-frequency acoustic source on the surface is activated and the reflected sound waves are recorded, typically, by tens of thousands of receivers. We term this process a *shot*, and it is repeated many thousands of times while the source and/or receivers are moved to illuminate different areas of the subsurface. The resulting dataset is dozens or hundreds of terabytes in size. The problem of transforming this dataset into an image is computationally intensive and can be solved with a variety of techniques. RTM is a high-end technique for generating images of the earth and is used in complex geologies to give detailed subsurface images.

The concept behind RTM is relatively simple. We start with a known earth model. This earth model might be simply acoustic velocity, but could also be anisotropic, elastic, or even visco-elastic. Scientists conduct two modelling experiments simultaneously through the earth model. Both attempt to simulate the seismic experiment conducted in the field, one from the perspective of the source and one from the perspective of the receivers.

The source experiment involves injecting our estimated source wavelet into the earth and propagating it from $t = 0$ to our maximum recording time $t_{\max}$, creating a 4D source field $s(x, y, z, t)$. Typical values for x, y, z, t are in the range 1000 to 10000. At the same time, we conduct the receiver experiment. We inject and propagate our recorded data, starting from $t_{\max}$ to t_0, creating a similar 4D volume $r(x, y, z, t)$. We have a reflection, where the energy propagated from the source and receiver is located at the same position at the same time and, thus, an image can be obtained by summing the correlation of the source and receiver wavefield at every time point and every shot:

$$i(x, y, z) = \sum_{\text{shots}} \sum_{t=0}^{t_{\max}} s(t, x, y, z) r(t, x, y, z) \tag{2.1}$$

For practical problem sizes, the 4D volumes can be large (many terabytes), which means that the storage and access of these volumes are key criteria for the performance of the method. RTM can be implemented using different algorithms that trade (re)computation for storage. Here we describe two algorithms that will be of use for HARNESS.

The most straightforward method of implementing RTM generates the full 4D volume of source data, stores it, then reloads while computing the receiver volume and performs the imaging cross-correlation. Figure 2.1 shows pseudo-code for this algorithm. The source wavefield data can be stored on disk or in memory, but the storage resource must have sufficient capacity and be fast enough to not unduly limit performance.

An alternative approach is to recompute the source wavefield, avoiding the storage of a 4D volume, but requiring up to 50% more computation than the storage approach. Figure 2.2 shows pseudo-code for this algorithm.

Table 2.1 compares the key characteristics of the two algorithms. The recomputation algorithm performs more computation and requires more memory (in order to compute on both source and receiver fields in parallel). However. it does not require any storage resources, while the storage-based scheme relies on a large volume of high-performance storage.

In theory, it is possible to choose between the different RTM algorithms on a per-shot basis depending on what is most efficient at a particular point.

For both algorithms, the core computational cost is modelling wave propagation. This is typically performed using 3D finite difference and, if hardware is available, this can be effectively computed on

```
model = load_earthmodel(...)
for t = 0 to tmax do
   add_stimulus(t, curr)
   propagate(curr, prev, model)
   if t mod image_step == 0 then
      store_field(t, curr)
   end if
end for
curr = zeros(nx,ny,nz)
prev = zeros(nx,ny,nz)
image = zeros(nx,ny,nz)
for t = tmax to 0 do
   add_receiver_data(t, curr)
   propagate(curr, prev, model)
   if t mod image_step == 0 then
      tmp = read_field(t)
      image += tmp*curr
   end if
end for
```

Figure 2.1: Pseudo-code for RTM algorithm: storing source wavefield.

```
model = load_earthmodel(...)
src_curr = zeros(nx,ny,nz)
src_prev = zeros(nx,ny,nz)
for t = 0 to tmax do
   add_stimulus(t, src_curr)
   propagate(src_curr, src_prev, model)
end for
swap(src_curr, src_prev) {Reverse time direction}
rec_curr = zeros(nx,ny,nz)
rec_prev = zeros(nx,ny,nz)
image = zeros(nx,ny,nz)
for t = tmax to 0 do
   add_receiver_data(t, rec_curr)
   propagate(rec_curr, rec_prev, model)
   propagate(src_curr, src_prev, model)
   if t mod image_step == 0 then
      image += src_curr * src_curr
   end if
end for
```

Figure 2.2: Pseudo-code for RTM algorithm: recomputing source wavefield.

	Store	Recompute
Compute cost	2	3
Memory capacity required	3N	5N
Storage capacity required	(NT/IS) * N	0
Storage throughput required	High	None

Table 2.1: Comparison of RTM algorithms. N is the spatial problem size ($NX * NY * NZ$), NT the number of propagation time steps, and IS the imaging step.

accelerator hardware such as an *field-programmable gate array* (FPGA)-based *data-flow engine*s (DFEs) or *general-purpose graphics processing unit*s (GPGPUs).

2.3 Industrial Practicalities of RTM

RTM is typically run on large clusters. Generally. run times of a few weeks are regarded as commercially acceptable, so a typical job could run for two weeks on 500 multi-core CPU nodes. A common problem dimension would be, for example, 10000 shots, where each shot is 1000x1000x500 spatial grid points x 5000 time steps. Memory requirements of 20-100GB per shot are common.

It is common to run one shot on one CPU node, using multiple threads to work together on the same problem, and then to run multiple shots on multiple CPU nodes. While it is possible to run the same shot across many machines, the cost of data exchange between the nodes can have a significant impact on performance and, thus, this is rarely used unless a node does not have sufficient memory to hold the data for a full shot. Accelerators such as a DFEs or GPGPUs are often used for RTM, and this can require multiple nodes to cooperate on each shot, since these devices (GPGPUs in particular) often have much smaller memory capacities than CPUs.

A key parameter affecting the run time of RTM is the seismic frequency. Increasing the modelled wavelet frequency requires decreasing the physical grid spacing and decreasing the time step spacing in order to maintain stability. Thus, run time is related to frequency as $O(f^4)$, and memory consumption is related to frequency as $O(f^3)$. Most commercial use of RTM is currently limited to low frequencies (e.g., 30Hz) by this cost. High-frequency RTM (e.g., 70-100Hz) is used more rarely.

2.4 Particular Characteristics Relevant to Cloud Computing

Seismic processing applications such as RTM are not commonly run in the cloud today. However, there is growing interest in exploring the opportunities this would provide. Often, RTM is run on behalf of an oil company by an oil service company. The service company will run the computation on its computers, but also provides the application that is running and the geophysics staff to manage the job, exercising quality control on the results.

Thus, there are two main potential customers for RTM: oil companies who own data running their own RTM computations on those data to produce images, and oil service companies running RTM on behalf of oil companies. In some cases, oil service companies may also run RTM on their own multi-client datasets.

Both of these customers can potentially benefit from using the cloud, but in different ways. Oil companies may run RTM infrequently, and thus may not want to maintain clusters in house. Thus, the cloud could be a good solution. Oil service companies will run large amounts of RTM, but may have spikes in demand that are hard to satisfy using in-house resources.

Demand for RTM in the cloud is therefore likely to be "lumpy", using large numbers of machines for short periods. In fact, the cloud could offer a benefit over in-house resources in allowing smaller latencies for an individual survey (e.g., using 7000 nodes for one day, rather than 500 nodes for two weeks). At the same time, RTM could potentially be used as a "back fill" job, utilising spare processing capability when it is available and reducing its resource load when there are other demands on the cloud resources—for example, RTM could run mainly at night, when demand for user-facing services is lower.

There are several major concerns when running RTM in the cloud that would need to be addressed:

1. RTM requires large volumes of input data in order to produce an output image (dozens of terabytes). Transferring these data over the Internet is not going to be practical in many cases, so hard drives or tapes would need to be physically shipped.

2. RTM is part of a larger data-processing workflow. To be efficient it may be necessary for the cloud to support all elements of this workflow, even if most are not as computationally demanding as the migration.

3. Seismic survey data is expensive to acquire and is highly commercially sensitive, making data security critical.

4. Legal restrictions may prevent the transfer of data outside certain geographical areas.

2.5 Opportunities to Exploit Heterogeneous Resources

RTM can benefit from heterogeneity of resources, particularly on the computation and storage sides. Since different versions of the core algorithm can be used, computation and storage can be traded against each other, allowing the optimum algorithm to be selected at a particular point in time given the resources that are available.

The ideal resources to be used for RTM will differ depending on the user's view of the trade-off between cost and overall job latency, and also the parameters of an individual job. Migrating a survey at low frequency may be quick to compute on a CPU, storing all data in *dynamic random-access memory* (DRAM), while migrating at higher frequencies may require the use of accelerators and either much larger and faster storage systems or use of the reverse propagation algorithm.

By providing a mix of heterogeneous resources and allowing a near-optimum system for a particular job to be rented for only the time required, a cloud solution could potentially even be cheaper than an in-house cluster with a configuration optimised for the most common case, but not ideal for jobs with other characteristics.

2.6 Specific Requirements

Overall, the HARNESS platform should be capable of optimising the RTM application. In particular, the following requirements need to be met:

R25	The HARNESS platform shall allow the expression of target latency or target cost	
Given the common use case of RTM, the cloud user will likely desire to specify either a cost target (i.e., run in the fastest time not exceeding this cost for the job) or a latency target (i.e., run in minimum cost, taking not longer than this amount of time).		
Task: T6.2, T6.3	**Innovation: medium** **Dependencies: None**	**Importance: moderate**

R26	The HARNESS platform shall support multiple algorithms	
The HARNESS platform shall be able to allow applications to be expressed with multiple choices of algorithm that could be used in their implementation. The platform can then choose among the algorithms to utilise at run time, based on the user's latency or cost target.		
Task: T3.1, T3.2, T6.1, T6.2	**Innovation: high** **Dependencies: R25**	**Importance: critical**

R27	The HARNESS platform shall exploit heterogeneous compute resources	
With multiple implementations of the core propagate function provided for different compute architectures, the HARNESS platform shall effectively exploit these architectures to meet the targets.		
Task: T3.1, T3.2, T6.3	**Innovation: medium** **Dependencies: None**	**Importance: critical**

R28	The HARNESS platform shall support fast linear read/write access to temporary data storage with understanding of duty cycle	
The storage-dominated RTM algorithm requires fast access to large amounts of temporary data storage. The application will need to have near-guaranteed read/write throughput for large linear accesses to a certain volume of storage. The platform should select the most appropriate storage medium automatically (DRAM, SSD, or disk). In addition, the platform must be able to understand the impact of the RTM access pattern (i.e., continuous write-read-erase cycles) on storage media with limited write cycles (such as SSD) and ensure that cost models reflect this.		
Task: T5.1, T5.2	**Innovation: medium** **Dependencies: None**	**Importance: critical**

2.7 Summary

RTM generates an image of the subsurface of the earth from seismic survey data. It is widely used in the oil and gas industry, but is not currently often run in cloud environments. Several significant challenges must be overcome to enable RTM to be effectively utilised in the cloud. However. the HARNESS platform can potentially expose major benefits of the cloud to RTM users, in particular the ability to complete a job under cost or time constraints and to exploit available heterogeneous resources.

3 In-Memory Databases: Delta Merge

The increasing demand for real-time business intelligence has caused a rethinking of how data are managed in large enterprises. Traditional data management systems are primarily designed for bulk batch processing of data, where the data are stored on massive disk arrays, not for the interactive responsiveness required of on-line data analytics.

In-memory databases have emerged as an alternative to disk-based databases. They embody new ways of organising data and, therefore, require new approaches to the implementation of access methods.

This chapter describes *Delta Merge*, one critical operation used to implement a high-performance database within the SAP HANA Cloud Platform and targeted by the HARNESS project.

3.1 General Domain Scope and Requirements

The SAP HANA Cloud Platform, shown in Figure 3.1, is an in-memory data management platform offering real-time performance for large data volumes with a current focus on business analytics. There are three main elements to the platform: (1) the SAP HANA AppServices, which is a set of shared services and capabilities provided by the application service in the platform allowing developers to create applications using multiple languages and frameworks (such as HANA, Java and other development services); (2) the SAP HANA DBServices, which provides developers with HANA database capabilities based on a cloud environment and a database service provided by the platform; and (3) the infrastructure layer, which provides the computation, communication and storage solutions required for hosting the platform depending on the deployment scenario.

The SAP HANA platform offers enterprises several opportunities.

1. **Make decisions in real-time:** HANA enables access to real-time analysis with fast and easy creation of ad hoc business statistics.

2. **Accelerate business processes:** HANA offers increased speed of information processes such as planning, forecasting, pricing, and offers.

3. **Unlock new insights:** HANA removes the constraints for analysing massive data volumes, trends, data mining and predictive analysis with the ability to work with structured and unstructured data.

4. **Improve IT efficiency:** HANA allows enterprises to manage growing data volume and complexity with lower cost of ownership.

Computation and storage requirements for the SAP HANA platform depend on the uncompressed database size, and thus CPU sockets and main memory size must scale accordingly. Compression rates vary depending on the data, but typically compression factors of 3-5 are common with higher values possible. For optimum performance, the compressed database should account for half of the main memory available. The SAP HANA platform license pricing reflects these scaling requirements,

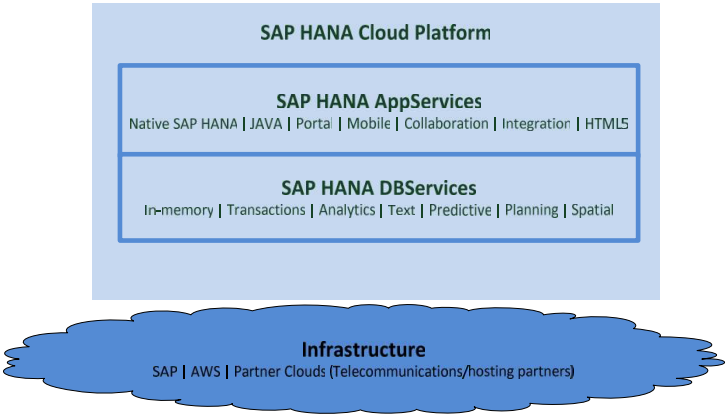


Figure 3.1: Elements of the SAP HANA Cloud Platform.

where licenses are based on in-memory storage capability, and range from extra small at 128 GB with a minimum requirement of two CPU sockets, to extra-large servers offering terabytes of main memory with eight CPU sockets required per terabyte.

A choice of deployment environments are available with variations in size, licensing model and deployment scenarios (on-premise, hybrid cloud and a full cloud solution). Current deployment solutions are:

- **SAP HANA One on Amazon Web Services.** The primary purpose of SAP HANA One is to provide a turnkey way for developers to inexpensively run use cases. It is a fully featured small instance of the SAP HANA platform that is available for production and commercial use. It provides 60 GB of main memory and is suitable for supporting ad hoc business intelligence or other applications that are designed to run in the cloud.

- **SAP HANA On Premise.** Working with certified hardware vendors, this offering enables customers to scale to desired levels. While this is typically a private cloud scenario, customers can also complement this option with a hybrid deployment with a SAP HANA cloud offering.

- **SAP HANA Enterprise Cloud.** The SAP HANA Enterprise Cloud (Figure 3.2) offering is a comprehensive cloud infrastructure combined with managed services. Customers can operate SAP HANA applications that run on top of the HANA platform in a managed cloud environment, e.g., SAP Business Suite and NetWeaver Business Warehouse. SAP HANA Enterprise Cloud is a fully scalable, enterprise-ready, mission critical, secure, and high-availability cloud offering with a fully managed-services approach.

The main differentiator between these deployment solutions is scale: HANA One offers limited scale, HANA on-premise offers any desired scale, and enterprise cloud provides cloud elasticity.

The current business model for cloud computing is based on leveraging commodity computation, communication and storage to provide low-cost application hosting. Economies of scale in cloud computing benefit large cloud vendors considerably. In particular, *capital expenditure* (CAPEX) costs can be considerably reduced as large volumes typically receive much discount, while hardware providers

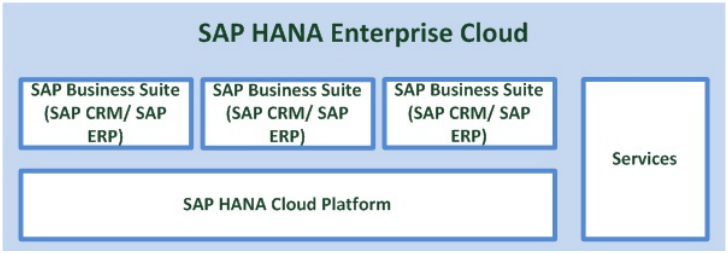


Figure 3.2: SAP HANA Enterprise Cloud.

will typically offer custom configuration based on this volume. On the other hand, *operational expenditure* (OPEX) costs (comparing a cloud-hosted solution to hosting the application on premise) are typically five to seven times lower [20]. Cloud vendors also offer high availability and a host of data centre security solutions.

While the limited size SAP HANA One on *Amazon Web Services* (AWS) can benefit from the advantages offered by cloud computing, since it can be readily implemented on commodity hardware, achieving similar benefits for the SAP Enterprise Cloud is much more difficult. The enterprise performance and elasticity requirements for Enterprise Cloud impose considerable technical requirements on the HANA infrastructure layer for high-end computation and main memory storage that break the economies of scale model that is the basis of cloud computing.

Heterogeneous computation, communication and storage solutions offer the opportunity to improve the performance and cost profile of infrastructure for the HANA Enterprise Cloud by deploying applications, or sub-tasks of applications, to these technologies. One sub-task that is ripe for deploying to such technologies is the *delta-merge* process. This process is a fundamental requirement of the SAP HANA in-memory database and is integral to maintaining the performance of the database.

3.2 Delta-Merge Application Description

The column store utilised in the SAP HANA database uses dictionary encoding and other compression algorithms to ensure that all relevant data are accessible in main memory. Figure 3.3 illustrates dictionary compression for a column of names. Dictionary encoding has the effect that a single write operation into this compressed data would require re-sorting and re-coding the compressed structure and, as such, a delta storage is used to maintain the list of write accesses to the data.

Figure 3.4 illustrates the data structures utilised to maintain a table within SAP HANA. Main Store contains two data structures: (1) Main Dictionary D_m and (2) Main Column C_m, which is a vector of value IDs. These value IDs are defined by the value position in D_m. Delta Store consists of an unsorted dictionary, Delta Dictionary D_d, Delta Column C_d, a vector of value IDs defined by value position in D_d, and Delta Map K_d, which maps the values to their positions in D_d.

Because each read of the database requires a search of both Main Store and Delta Store, Delta Store needs to be organised in such a way that it does not limit the read performance of the system. This is accomplished using M_d, enabling a value ID entry to be found in $\log(|D_d|)$. The map is implemented as a cache-sensitive B+ tree (CSB+ tree) and is therefore ordered by the dictionary value that maps to its respective value ID (its position in D_d).

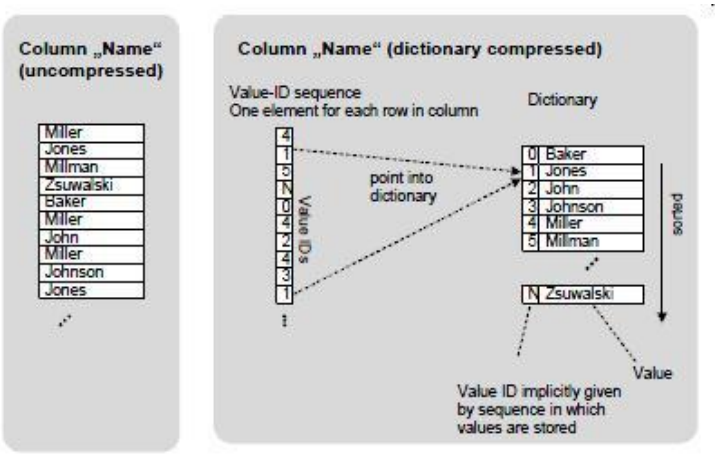


Figure 3.3: Column-store dictionary compression.

D_m and D_d are not related, resulting in possible repetitions of values in each. However, each individual dictionary will only contain unique values. Each update of Delta Store requires a look up for the value in the M_d; if the value does not exist in the map, then the dictionary value is appended to the end of D_d and the associated index value is appended to the end of C_d. If the value does occur in M_d, then the associated value ID, from M_d, is appended to C_d, while no operation occurs on D_d.

The consequence of maintaining two data structures is the need to occasionally merge them. This preserves the read and write performance by ensuring that Delta Store does not grow overly large, but it requires additional overhead to perform this merge process.

The delta-merge process requires two main steps, firstly merging D_M and D_d, and secondly updating C_m and C_d. Once C_m and C_d are updated, C_d is simply appended to the end of C_m to create the new Main Store column C_n. As the SAP HANA system must continue to operate while the delta-merge process is performed, it is necessary to permit reading of an old Main Store and Delta Store during a merge, as well as support writing and reading to a new Delta Store structure that is generated after a merge operation begins, as illustrated in Figure 3.5.

3.3 Industrial Practicalities of Delta Merge

Delta merge frequencies are highly dependent on the types of transactions and the frequency of transactions the SAP HANA database must handle. Systems where there are a high number of *online transaction processing* (OLTP) transactions will also have a large delta-merge load, since these transactions involve updates to the database that are inherently handled by the delta storage layer.

Typically, these systems would have high data volumes, in the range of terabytes, and transaction rates of hundreds per second. An example are the point-of-sale transactions of a large European retailer. The delta-merge process can account for a non-trivial amount of system resource usage on a SAP HANA appliance, but the exact amount will depend on the customer configuration and the nature of the data and its usage.

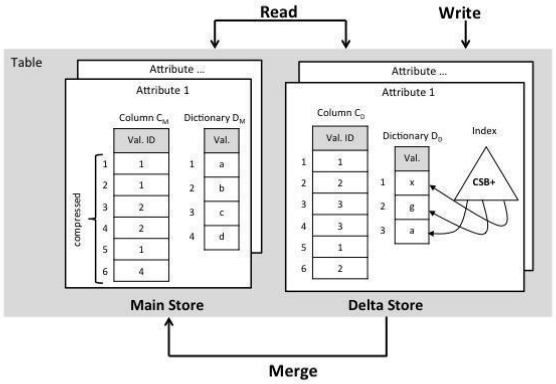


Figure 3.4: SAP HANA table data structures.

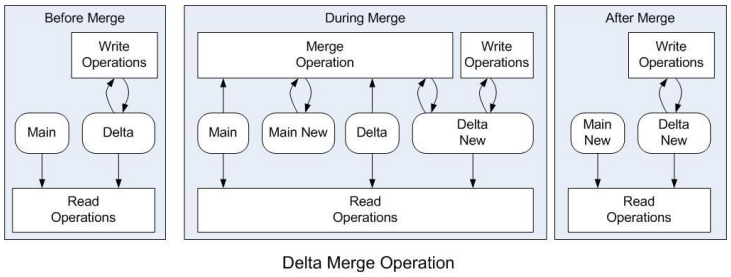


Figure 3.5: SAP HANA Delta-Merge operation.

3.4 Particular Characteristics Relevant to Cloud Computing

Delta merge is not a process that runs ad hoc within a traditional cloud environment. In particular, delta merge currently runs within SAP HANA appliances on SAP Enterprise Cloud. Within traditional commodity-based cloud infrastructures, such as AWS, delta merge can run on top of traditional commodity resources.

SAP HANA database tables typically can consist of 100 to 300 columns, with a narrow subset of attributes being continually accessed. Delta merge operates at the table level and, when triggered, all columns within that table will require merging. All columns have no data dependencies and, therefore, can be merged in parallel if enough resources are available. Typically, customer data will follow the Pareto principle, with the majority of columns consisting of small dictionaries [21], but the minority of columns will account for the largest portion of resource usage on the SAP HANA appliance.

Not all SAP HANA instances require offloading the delta-merge functionality to accelerators. However for customers with high transaction volumes, offloading to accelerators frees up traditional CPU resources to handle analytic and transactional queries. In addition to the reduced OPEX costs,

offloading delta-merge functionality to accelerators enables better throughput at lower frequency and power usage, and also reduces the running time of the function per table, enabling opportunities to optimise the read and write performance of the database.

While delta merge has a number of internal triggers for when columns should be merged, SAP HANA also provides a mechanism to begin a table merge using a standard SQL call. This capability enables a large degree of flexibility in managing the merge process in a cloud environment. For example, when accelerator nodes are free, merges that may not trigger until a later point can be started in an ad hoc fashion.

Within a heterogeneous cloud environment the expected goals are:

- when required, the platform should provide more CPU resources for the SAP HANA appliance to handle traditional transactions by offloading delta-merge functionality to accelerators;
- offloading the delta-merge process to cloud-based accelerators should lower its running time per table and increase the transactional rates that the SAP HANA appliance can handle.

Because the delta-merge algorithm is fundamental in the management of data within a SAP HANA database, this constrains how the process is run within a cloud environment, requiring accelerators to be coupled with high-speed read and write access. There are a number of issues that will need addressing to manage the delta-merge process within a cloud environment:

- high performance coupling of accelerators to SAP HANA appliances;
- intelligent scheduling of the delta-merge process based on column data type and size; and
- efficient management of the delta-merge process with a need to manage the parallel execution of table merges, and the parallel execution of merges within a table across multiple technologies.

While the streaming nature of the algorithm lends itself to accelerator-based offloading, high-speed data transfer to the accelerators is fundamental to speed up the algorithm. Typically, this should support read and write speeds on the order of gigabytes per second.

3.5 Opportunities to Exploit Heterogeneous Resources

Heterogeneous resources provide many opportunities to optimise the delta-merge process, both per table and across the entire database. Offloading computations to accelerators enable opportunities to optimise the read and write performance of the database and the capability to merge different columns based on the data transfer requirements. Typically, larger *peripheral component interconnect express* (PCIe) data transfers achieve higher throughput. Therefore, any optimisation of the table merge may involve merging large columns on accelerators, while smaller columns may be better handled by local CPU resources. In addition, different accelerator implementations have various computational performance characteristics: depending on resource usage and performance requirements, a mix of resources may be chosen.

The delta-merge process moves data from a data structure that handles reads and writes (delta storage) to a structure that is highly optimised for reading (main storage). After the process, both the

read and write performance of the database will be improved as the delta structure will be minimal and have an improved read and write performance. Therefore, having the ability to perform this process on accelerators enables opportunities to further optimise the performance of the database, and not interfere with analytical queries that are ongoing by using SAP HANA appliance resources.

3.6 Specific Requirements

Overall, the HARNESS platform should be capable of optimising the Delta-Merge application. In particular, the following requirements need to be met:

R29	**The HARNESS platform shall provide support for specification of particular resources**
Since the delta-merge functionality requires close coupling of accelerators as well as specific hardware resources on the accelerators, the HARNESS platform must enable the specification of accelerators based on local memory and hardware requirements.	
Task: 6.1, 6.2	**Innovation: medium Importance: critical** **Dependencies: R3, R9**

R30	**The HARNESS platform shall support multiple application implementations**
The HARNESS platform must be able to support the delta-merge process for different data types that may have differing implementations and vary across a table. As all data types may not be supported across the accelerators, the platform must then choose which implementation to utilise at run time.	
Task: 3.2, 6.3	**Innovation: high Importance: critical** **Dependencies: R1, R2, R9, R11, R12**

R31	**The HARNESS platform shall efficiently schedule small compute jobs**
The HARNESS platform should be able to execute column-based merges and the execution decision (on which accelerator to execute the merge) should have negligible impact on the table-merge running time. Since many column merges may be less than one second, the decision process must account for a small percentage of this compute time.	
Task: 3.2	**Innovation: high Importance: critical** **Dependencies: R1, R2, R9, R11, R12, R13**

R32	**The HARNESS platform shall intelligently schedule jobs to resources**
The HARNESS platform should be able to decide on the optimal location to run merges depending on resource availability of the host, the accelerators, data transfer times to accelerators and any requirements of the delta-merge process. An example requirement may be the column length, which affects the data-transfer requirements. Merges may be scheduled in parallel or sequentially based on available memory and current system demand.	
Task: 3.2, 6.2, 6.3	**Innovation: high Importance: critical** **Dependencies: R1, R2, R9, R11, R12, R13, R14**

R33	The HARNESS platform shall support elastic resource allocation
The HARNESS platform should be able to adapt to changes in application demands and increase or decrease accelerator resources when required.	
Task: 3.2, 3.3, 6.2, 6.3	**Innovation: high Importance: critical** **Dependencies: R1, R2, R9, R11, R12, R13, R14**

3.7 Summary

Economies of scale that are the mainstay of traditional cloud computing vendors are difficult to achieve for the SAP HANA Enterprise Cloud, which requires high-performance and custom main-memory configurations. Heterogeneous technologies provide an opportunity to improve the cost and performance profiles of the necessary infrastructure by deploying applications, or sub-tasks of applications, to these technologies. Once such sub-task is the delta-merge process, which is fundamental for maintaining the high performance of the SAP HANA database, but consumes traditional SAP HANA appliance resources that could be best used for analytics. There are a number of technical requirements that need to be addressed to accelerate this process, but achieving these within the HARNESS platform can provide significant benefits for the SAP HANA Enterprise Cloud, in particular, and the in-memory database domain, in general.

4 Data-Flow Computing: AdPredictor

AdPredictor represents a class of modern industrial-scale applications, commonly known as *recommender systems*, that target either open-source or proprietary on-line services. For example, one such service is Mendeley [17], a free reference organiser and academic social network that recommends related research articles based on user interests. Another service is Bing [1], a commercial on-line search engine that recommends commercial products based on user queries. In general, items are matched with users and, due to today's data deluge on either data set, these computations are usually run on large-scale data centres.

This chapter describes *AdPredictor* [10], a Bayesian machine-learning system that is used by Bing for advertisement ("ad") recommendations. It does so without loss of generality, as such systems are content agnostic and, therefore, easily portable to other applications and languages, sharing similar challenges at scale.

4.1 General Domain Scope and Requirements

Many data-parallel computing frameworks express computation tasks as vertices and the data I/O relationships among those tasks as edges in a directed (usually acyclic) graph, a model commonly known as *data-flow computing*. In Google's MapReduce [5, 6], arguably the most popular among these frameworks, the data movement among tasks is constrained by the *map*, *shuffle* and *reduce* model of computation, depicted in Figure 4.1. In later systems (e.g., Microsoft's Dryad [14]), users can specify arbitrary task graphs with fewer constraints. In fact, the MapReduce model has spurred numerous research groups, both in academia and in industry, to build various large-scale data-flow computing engines—each adopting different strategies for computation, communication, and storage resource management—in order to facilitate inferences over the sheer volume of information produced daily in commercial data centres.

Microsoft's Bing search engine, for example, is visited by hundreds of million users, generating approximately 12GB-15GB of user search and click logs daily [10]. As we describe below, swift and accurate processing of these click observations is paramount in generating revenue for Microsoft, as well as other major Web indexing service providers such as Google, Yahoo!, and Facebook.

AdPredictor was developed at Microsoft Research Cambridge and is now responsible for 100% of paid search traffic in Microsoft's Bing search engine. As such, our description of AdPredictor is primarily based on information published by Microsoft. Their proposed inference model, however, exemplifies a class of distributed machine learning algorithms and is applicable to a wide range of real-time, large-scale distributed inference systems [4, 9, 19].

Intuitively, *click-through rate* (CTR) calculations in parallel work as follows. Given a stream of empirically observed ad displays, either clicked or not clicked, what is the probability that the next displayed ad will be clicked during a user search session? Complex data inference systems, where neither the exact dimensionality of the observable parameters nor their in-between relationships are

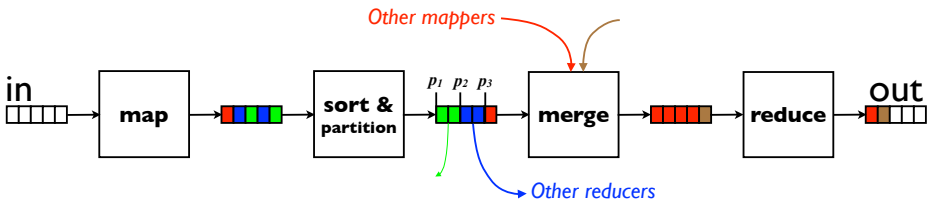


Figure 4.1: The MapReduce model. The input data are split into shards and delivered to identical *map* tasks, each running on a processor core. Every map task generates a number of key/value pairs that are first sorted, then partitioned, and finally transmitted over the network in such a way that intermediate results from different map tasks with the same key land on the same processor and then merged together. This is known as the *shuffle* phase. For each group of keys, a *reduce* task is responsible for aggregating (e.g., multiplying or adding) their values.

certain, require precise *a posteriori* revisions of any prior beliefs in light of new observations. According to Bayes' theorem, the posterior belief $p(\theta|X)$ is the likelihood function $p(X|\theta)$ of a new observation X given parameters θ, times the prior probability distribution $p(\theta)$. It has the form:

$$p(\theta|X) = \frac{p(X|\theta)\ p(\theta)}{p(X)},$$

where the denominator $p(X) = \int p(X|\theta)\ p(\theta)\ d\theta$ is a normalising factor [2]. It is not possible to calculate the exact posterior beliefs for parameters θ, thus AdPredictor resolves to an approximate solution. In particular, it adopts a model, known as a *factor graph*, that essentially allows the sharding of X and θ; thus, computation of $p(\theta|X)$ can be executed in parallel by map tasks, and their results aggregated (multiplied together) by reduce tasks.

Before delving into the opportunities that arise from heterogeneous data-flow computing, in general, and from the HARNESS computation, communication, and storage technologies, in particular, the next section first places AdPredictor in the context of data-inference tasks.

4.2 AdPredictor Application Description

AdPredictor predicts *sponsored search* results. Sponsored search works as follows. When a user submits a query to a commercial search engine (e.g., Google or Bing) the engine retrieves an ordered list of relevant ads in addition to the relevant link information answering the user's search query. Ads are usually displayed at the top of the search results page or at the sidebars (Figure 4.2). On one hand, advertisers compete by biding for certain query terms, as it is in their interest to appear first in that list, and pay on a per-click basis. On the other hand, companies like Google or Microsoft rank relevant ads according to their CTR, as it is in their interest to generate more revenue by displaying ads that will be most likely clicked, and charge in such a way that advertisers have no incentive to lower their bid. AdPredictor is responsible for calculating an ad's CTR—in other words, the probability for an ad to be clicked during a user search session.

Observations about ads, also termed *ad impressions*, are tuples X that consist of attributes x_i related to a particular user search session. Example attributes include the user's identifier, the IP address, the

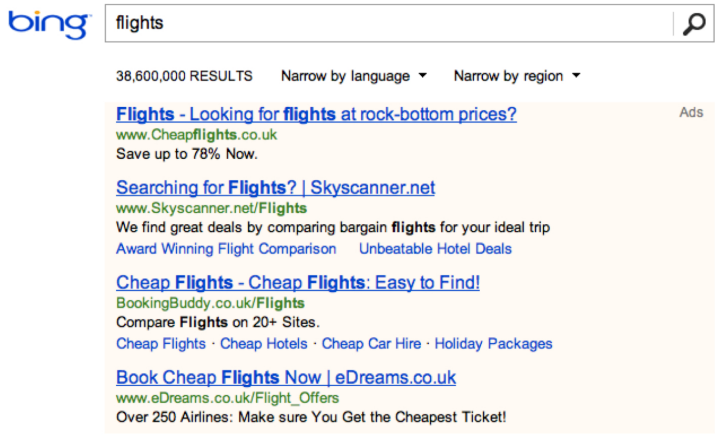


Figure 4.2: Bing's sponsored search results for query "flights", retrieved on 23 May 2013. Ads are displayed atop 38.6 million organic search results.

Ad features	bid keywords or phrases, ad identifier, ad title, ad description, landing page URL, advertiser identifier, ad campaign, ad group
Query features	search keywords, query expansion terms
Context features	display location, display order, number of ads displayed, geographic location, time, user search history, IP address, user age, user gender

Table 4.1: Basic input features.

query's terms, and the ad's relative ranking on the page. Table 4.1 presents a summary. For each ad impression, the system observes whether the ad was clicked or not clicked. These data streams are stored in log files. AdPredictor processes $O(10^9)$ such ad impressions per week [10].

Every attribute x_i is either 0 or 1. Features, such as a user's id, can be viewed as sparse binary vectors of length m, and the system maintains distinct prior beliefs for each feature value, say, users $u_1, \ldots, u_m$. For example, the user id feature has values:

$$\begin{aligned} u_1 &= 1, 0, 0, 0, 0, \ldots; \\ u_2 &= 0, 1, 0, 0, 0, \ldots; \\ &\vdots \\ u_m &= \ldots, 0, 0, 0, 0, 1 \end{aligned}$$

In practice, $x_i = k > 0$ denotes a sparse binary vector of $m \geq k$ bits, all set to 0 but the k^{th} value.

The remainder of this section describes the two main inference tasks of AdPredictor, namely *online training* and *online prediction*.

4.2.1 Online training

Given an observation (X, y), where $y = 1$ when an ad was clicked and $y = -1$ otherwise, and prior probability distribution $p(\theta)$, the system must infer the new posterior $p(\theta|X, y)$. For every feature value x_i, AdPredictor maintains a Gaussian belief over parameter θ_i. Assuming that parameters θ_i are *independent and identically distributed* (i.i.d.) random variables, then the prior probability distribution of the system is:

$$p(\theta) = \prod_{i=1}^{n} \mathcal{N}(\theta_i; \mu_i, \sigma_i^2)$$

In practice, these prior beliefs are stored in memory as vectors:

$$\boldsymbol{\mu} := (\mu_1, \mu_2, \ldots, \mu_n)^{\mathrm{T}}$$

and

$$\boldsymbol{\sigma^2} := (\sigma_1^2, \sigma_2^2, \ldots, \sigma_n^2)^{\mathrm{T}}$$

The likelihood function $p(y|X, \theta)$ is then approximated using expectation propagation on factor graphs in order to derive the following closed-form update equations for the mean and variance of the posterior distributions $\mathcal{N}(\theta_i; \widetilde{\mu}_i, \widetilde{\sigma}_i^2)$:

$$\widetilde{\mu}_i = \mu_i + y x_i \cdot \frac{\sigma_i^2}{\mathcal{S}} \cdot v\left(\frac{y \cdot \sum_{j=1}^{n} x_j \mu_j}{\mathcal{S}}\right)$$

and

$$\widetilde{\sigma}_i^2 = \sigma_i^2 \cdot \left[1 - x_i \cdot \frac{\sigma_i^2}{\mathcal{S}^2} \cdot w\left(\frac{y \cdot \sum_{j=1}^{n} x_j \mu_j}{\mathcal{S}}\right)\right]$$

In the equations above, $\mathcal{S}^2 = \sum_{j=1}^{n} x_j \sigma_j^2 + \beta^2$ is the sum of variances of the i.i.d. parameters involved in the training instance X, plus the variance β^2 of the Gaussian noise process that generated it, and $v(\cdot)$ and $w(\cdot)$ are functions that control the magnitude of the mean and variance update, respectively, as illustrated in Figure 4.3.

Mean and variance updates highlight the following three intuitions behind AdPredictor, also illustrated in Figures 4.4(a), 4.4(b), 4.4(c) respectively:

(a) Every observation leads to a reduction in variance. Positive observations (clicks) increase the mean value of the parameters involved, whereas negative observations (no clicks) decrease them.

(b) The magnitude of an update for a parameter θ_i is proportional to its variance, normalised by its contribution to the overall uncertainty of a prediction (e.g., $\sigma_i^2/\mathcal{S}$ for means). Intuitively, if there is a lot of uncertainty about a parameter, its mean increases or decreases in large steps, whereas if a parameter is tightly constrained the update step is small.

(c) When argument $y \sum_{j=1}^{n} x_j \mu_j < 0$, then function v (respectively, function w) evaluates to a large number and the update step is large; otherwise, the step is small (see Figure 4.3). Intuitively, in the former case the system was "surprised" by an observation X.

The aforementioned posterior parameters $\tilde{\boldsymbol{\mu}}$ and $\tilde{\boldsymbol{\sigma}}^{\boldsymbol{2}}$, calculated for a given observation (X, y), are used as priors for the next observation (X', y'), and so on, naturally leading to an online learning system.

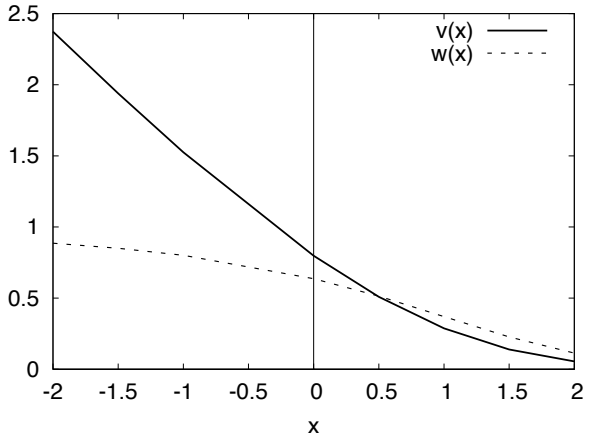


Figure 4.3: Learning step size functions $v(x)$ and $w(x)$.

4.2.2 Online prediction

Given posterior parameters $\boldsymbol{\mu}$ and $\boldsymbol{\sigma^2}$, then the probability of click ($y = 1$) or no click ($y = -1$) for an input X is:

$$p(y|x) = \Phi\left(\frac{y \cdot \sum_{j=1}^{n} x_j \mu_j}{\mathcal{S}}\right)$$

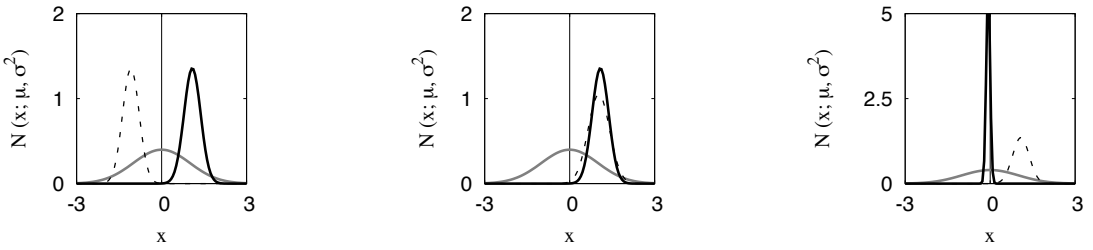
where $\Phi(t) = \int_{-\infty}^{t} \mathcal{N}(s; 0, 1)ds$ is the cumulative Gaussian density function such that $p(y|x) \in [0, 1]$.

4.3 Industrial Practicalities of AdPredictor

The aforementioned inference tasks, training and prediction, raise three major practical issues in a real-world ad delivery system. The first two issues are related to the quality of predictions and illustrate the effects of the system's (in)accuracy on user experience. The third issue is related to the system's scalability and reflects on the anticipated capabilities of the HARNESS cloud platform. Overall, these three examples demonstrate the subtle trade-off between accuracy and performance in recommender systems, further elaborated in Sections 4.4 and 4.5.

4.3.1 Adversarial behaviour

Consider the scenario of Figure 4.4. Users a and b are two extreme cases of adversarial behaviour, and they can be attributed to *bots*, software processes that perform automated tasks on the Internet. For example, a process that always clicks on ads can be used to deplete an advertiser's budget; a process that never clicks ads can be used to steal search query results and populate them with their own ads. These processes must be detected and reported, and the system must avoid charging advertisers or biasing its beliefs due to such behaviour.



(a) Posterior distributions after 100 click observations for user a (black) and 100 no-click observations for user b (dashed).

(b) The posterior distribution for user a after 10 click observations (dashed) and after 100 click observations (black).

(c) After 100 click observations for user a (dashed), the system observes a no-click event. The 101^{th} event surprised the system (black).

Figure 4.4: An example training scenario. AdPredictor observes only two users a and b, represented as vectors $X_a = (1, 0)^{\mathrm{T}}$ and $X_b = (0, 1)^{\mathrm{T}}$, respectively. User a always clicks ads, while user b never clicks them. AdPredictor initially assigns $\mathcal{N}(x; 0, 1)$ to each parameter (grey line). That is, predicting whether a user will click on an ad or not is based on "a flip of a coin". The more observations X_a and X_b are processed, the more accurately the system predicts that user a always clicks, while user b never clicks.

4.3.2 Exploration of ads

As discussed in Section 4.2, the more data about a particular parameter the system observes, the smaller its variance. Thus, the system predicts which ads to display with higher confidence. But the system's predictions also determine the future training set of AdPredictor. This means that there is a large pool of ads that are less and less likely to be impressed on users, and therefore clicked, which also has an effect on the overall performance of the system. In practice, systems like AdPredictor need also explore ads with low CTR.

4.3.3 Memory footprint

Users, URLs, and queries are all Zipf distributed (heavy-tailed) on the Web, making the model size—i.e., the length of vectors $\boldsymbol{\mu}$ and $\boldsymbol{\sigma^2}$—unnecessarily large. Consider, for example, feature values for user ids or IP addresses. Eventually, the system will observe billions of them, increasing the memory footprint of the training and prediction engine, $\boldsymbol{\mu}$ and $\boldsymbol{\sigma^2}$. But most feature values are infrequent, usually appearing only once and, therefore, the system can prune them. A feature can be pruned if its contribution in predicting $p(y|X)$ is small. In practice, features are binned together (e.g., user ids according to demographics, IP addresses according to subnets) so that when a pruned feature reappears, the prior belief for that parameter is already biased.

This contextual sharding of the parameter space θ, together with the inherent data parallelism of the stream of click/no-click observations (X, y) along the time axis, form the basis for a Web-scale implementation of AdPredictor described in the following section.

4.4 Particular Characteristics Relevant to Cloud Computing

Millions of ads, billions of users, billions of daily ad impressions: these are but a few of the scale characteristics of AdPredictor and related systems in a production environment. The prediction system must have a small *random-access memory* (RAM) footprint to be able to serve ad requests with a response time of just a few hundred milliseconds [10, 4]. In turn, accurate CTR prediction requires a fast, parallel training algorithm. The remainder of this section places AdPredictor's Bayesian inference model in the context of MapReduce computations.

Recall the Bayes theorem:

$$\text{posterior} \propto \text{likelihood} \cdot \text{prior}$$

Given a collection of k ad impressions $\mathbb{X} = X_1 \dots X_k$ and their corresponding click or no-click events $\mathbb{Y} = y_1 \dots y_k$, the computation of the posterior can be factorised because the data set $\mathbb{X}$ is i.i.d.:

$$p(\theta|\mathbb{X}, \mathbb{Y}) \propto \prod_{i=1}^{k} p(Y_i|\theta, X_i) \cdot p(\theta)$$

From a system's perspective, this means that the computation of the posterior can be carried out by independent *map* tasks, each parsing data shards $X \in \mathbb{X}$, $y \in \mathbb{Y}$, and their results can be combined together by *reduce* tasks, by multiplication of the corresponding likelihoods.

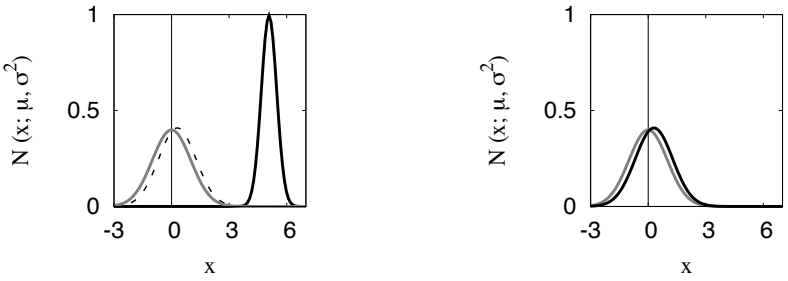
There are two main problems that need to be addressed in a parallel implementation of AdPredictor. The first one relates to the quality of the approximations and the second to the shared state requirements of the system.

4.4.1 Quality of predictions

As described in Section 4.2.1, AdPredictor's online training algorithm works sequentially: a training instance X_i uses the posterior updates calculated for a previous instance X_{i-1}. However, this is no longer the case in a parallel implementation of the training algorithm. This problem is inherent in similar online machine learning algorithms, and the effect is that the approximated probability distribution diverges from the actual one. Figure 4.5(a) illustrates this behaviour. The solution is to dampen the effect of each parallel update by a normalising factor that is proportional to the size of the data shard each map task processes. The effect is illustrated in Figure 4.5(b). While a sequential run provides the ground truth to evaluate any parallel solution, it is also prohibitive either due to large data sets or to end-to-end completion-time constraints.

4.4.2 Shared state

The model parameters $\boldsymbol{\mu}$ and $\boldsymbol{\sigma^2}$ are shared across all map tasks. Given data variables x_i, a map task can determine which model parameters are required to get (read) and, subsequently, set (update). In a MapReduce computation, this shared state needs to be transferred to and from the individual map tasks. In practice, click observations are usually first aggregated and processed on a *per user* or a *per advertiser* basis in order to also partition and distribute the corresponding model parameters θ to the machines where the computation takes place. This correspondence between data variables (e.g., user groups) and



(a) Sequential (dashed) vs. parallel (black) training of AdPredictor. The posterior distribution computed by the latter diverges from the actual posterior distribution.

(b) By normalising the updates produced by each parallel thread according the size of their data shard (1%), the resulting posterior distribution matches that of a sequential run.

Figure 4.5: A 100-dimensional biased click observation $\{x_i = 1\}_{i=1}^{100}, y = 1$ and a prior distribution $\mathcal{N}(x; 0, 1)$ with mean 0 and variance 1 (grey). A sequential implementation runs for 100 iterations and the results are compared with a parallel version, where each of the 100 iterations runs in parallel.

machines is constantly changing, either because of machine failures or because the corresponding model size grows large enough to signify a further re-partition of the data.

Based on these Web-scale characteristics of AdPredictor, the next section summarises the storage (Section 4.5.1), communication (Section 4.5.2), and computation (Section 4.5.3) tasks of an AdPredictor's MapReduce implementation for the HARNESS platform (Section 4.5.4).

4.5 Opportunities to Exploit Heterogeneous Resources

In the context of HARNESS, we aim to further enhance data-flow graphs by taking into account the different capabilities of heterogeneous resources. Without loss of generality, we are building a version of AdPredictor that consists of the following four phases:

1. *parsing* the user-click logs, filtering out any irrelevant information;
2. *sorting* the key/value pairs generated in the previous phase;
3. *aggregating* the sorted pairs per user or advertiser key; and
4. *training* AdPredictor's model with the aggregated results.

4.5.1 Data sharding

The first three phases are I/O-bound, since data must be read off disks, parsed, filtered, and finally sorted and aggregated by the user's (or advertiser's) key. Filtering is used to exclude attributes from the data set that do not contribute to AdPredictor's inference model. Sorting and aggregating based on a specific key is used to build inference sub-models that contribute to more accurate predictions. User id and/or advertiser id are two such attributes, since AdPredictor aims to improve the user experience by selecting

the most relevant ads, as well as increase revenue by selecting the highest grossing advertisers. In light of these I/O-intensive tasks, the opportunities for heterogeneous storage can be summarised as follows.

Recall from Sections 4.3.3 and 4.4.2 that a run of AdPredictor can parse billions of click events to generate an inference model that can exceed the memory limits of a single server. Thus, based on hints provided by developers on the expected amount of data in and out of AdPredictor, the provider can devise an appropriate partitioning of map tasks to ensure that the derived data shards can fit onto the fast disks and memory of servers. When multiple jobs are competing for resources, since these phases are I/O-bound, the provider can derive a coarse-grained approximation of the time required to run them depending on the amount of memory available and the disk read/write throughput. This information can be leveraged to efficiently assign resources to a job while ensuring that job deadlines are not missed.

4.5.2 In-network processing

To each data shard there corresponds a partition of the inference model that must be read in memory to be updated—at least those parameters for which the system holds *a priori* belief other than the default. Locally updated models are then merged to create a (hierarchical) global model. Updates are computed by map tasks and emitted over the network to be aggregated by reduce tasks. Let us momentarily leave computation aside (until Section 4.5.3), and focus on in-network data reduction.

The intermediate results generated by map tasks, potentially hundreds of GBs of data, must be transferred to the servers responsible for running the aggregation function (addition and/or multiplication). This stage is typically network bound: depending on the number of aggregation servers used, the bottleneck could be either the scarce capacity in the core of the network, since data centres are usually heavily oversubscribed [11], or the limited inbound bandwidth of the servers (typically, 1 Gbps). In light of these limitations, the following opportunities arise.

For partially reducing the network traffic, systems like MapReduce or Dryad allow users to define an additional function, namely a *combiner*, to aggregate the local data generated by a server [5, 22]. In HARNESS, this functionality can be extended by running the combiner function within the network, taking advantage of the computation capabilities of modern switches and routers [3, 7, 12, 18]. This action action can significantly reduce the network traffic (and, consequently, the end-to-end completion time) because data from multiple servers is aggregated at multiple levels of a data centre's fat-tree topology.

4.5.3 Iterative training

Tasks can be mapped to accelerator-specific code, such as GPGPU kernels and/or FPGA-based DFE kernels. In computing the new posterior distributions, for example, a map task may iterate more than once over a particular data shard. Once data are copied to an FPGA's DRAM, however, such a task can benefit greatly from its high-bandwidth memory access channels (ideally, 40GB/s) and inherent data parallelism.

The training phase of AdPredictor is an iterative, computationally intensive process, not only at the level of individual map tasks, but also at the level of a MapReduce run. Once data are sharded, partial updates computed, and the posterior distributions of model parameters aggregated, the process is usually repeated for a number of iterations, until the model converges. This is required in order to resolve any data dependencies that are not *a priori* available to individual map tasks, and emerge when individual

data shards share a portion of the model parameter space. In that case, the output of the first MapReduce iteration is the input to the second, and so on. The higher the degree of parallelism, the less accurate the model computed in a single iteration, the more iterations are required for AdPredictor to converge to the exact posterior distributions. This trade-off between accuracy and speed of convergence gives rise to the following opportunity for the HARNESS platform.

4.5.4 Building blocks

Developers can use a domain-specific language to specify the application's functional and non-functional requirements, for example by using high-level building blocks made available by the HARNESS platform, as in Merge [16, 15] or Infer.NET [13]. These components can be automatically mapped to accelerator-specific code that runs either at the edges or in the core of a data centre's network. The platform can then decide *what* accelerated version of the code to use, *when* to schedule the computation, and *where* the corresponding data shards will reside based on the time and budget constraints of the tenants and the current availability of hardware accelerators.

The processing order of individual updates affects the system's accuracy and speed of convergence (see Section 4.2.1). Thus, sorting and merging tasks during one or more shuffle phases of MapReduce play an important role in resolving data dependencies (i.e., *what*, *when*, and *where* scheduling constraints) amongst map and reduce building blocks.

4.6 Specific Requirements

Overall, the execution of AdPredictor on the HARNESS platform should complete faster than a CPU-only implementation that runs, say, on a homogeneous cluster of machines without accelerated computation, communication, or storage components. The main performance benefits are expected to come from HARNESS mechanisms that efficiently exploit such heterogeneous resources in a well-defined and self-contained data-flow computing platform. The following set of requirements highlight the anticipated results:

R34	**The HARNESS platform shall support coarse-grained, resource-agnostic, goal-centric specifications of inference tasks**
Application-specific choices (e.g., keys to group by data, coarse-grained data dependencies between computation tasks, accuracy of results, and expected completion time) should be expressible in the application manifest language of the HARNESS platform. This will give programmers the freedom of expressing complex inference tasks based on high-level goals; and providers the freedom of realising the specified data flow model with the most efficient software and hardware assembly.	
Task: T6.4	**Innovation: medium Importance: moderate** **Dependencies: R1–R8**

R35	The HARNESS platform shall efficiently schedule inference tasks on heterogeneous processors
With readily available accelerated application building blocks targeting, say, FPGA or GPGPU kernels, an AdPredictor run should complete faster than a CPU-only distributed implementation while satisfying data dependencies and precedences over updates.	
Task: T3.1–T3.3	**Innovation: high** **Importance: critical** **Dependencies: R10**

R36	The HARNESS platform shall support in-network aggregation
The use of in-network processing elements, such as programmable switches, should stem from the application specification and, in particular, the derived data-flow graph. The hierarchical, in-network data reduction tree provided by HARNESS should consume less bandwidth than a default centralised reducer.	
Task: T4.3	**Innovation: high** **Importance: critical** **Dependencies: R15–R19**

R37	The HARNESS platform shall support multiple data-storage access patterns
Parsing, sorting, and partitioning are data I/O-intensive tasks that should benefit from fast disks and custom read/write (get/set) abstractions built atop a distributed storage system. For example, the iterative nature of AdPredictor's training phase can benefit from storing results in memory.	
Task: T5.1–T5.3	**Innovation: high** **Importance: critical** **Dependencies: R20-R24**

4.7 Summary

AdPredictor, following the process used by Google, Facebook, and Microsoft, generates a Bayesian prediction model (a network of data and parameter factors) that is used to select which advertisements to impress upon users along with the results of a user search query. AdPredictor is a particular instance of recommender systems that aim to improve both company revenue, as well as the user experience.

AdPredictor is an online algorithm. That is, new observations update previously held beliefs about users, advertisers, and other features related to sponsored (or paid) search. The more observations the application observes, the more accurate are its predictions. In order to process the sheer bulk of data generated every day by users, AdPredictor must run in parallel. But in doing so, careful consideration must be given to data sharding, storage, and aggregation to best resolve any circumstantial data dependencies and achieve high degrees of parallelism.

By incorporating heterogeneous resources into the HARNESS cloud platform, it is now possible to innovate in accelerating data-flow graph applications at the coarse grain of map and reduce tasks, and associated data movements, within a data centre network, as well as at the fine grain of data-flow computing with FPGA and GPGPU co-processors, where individual map and reduce tasks can be further accelerated.

5 Summary of Platform Requirements

The requirements of the three industrial use cases have been detailed in Sections 2.6, 3.6 and 4.6. These requirements have been explained from the viewpoint of the application supplier and, as such, the commonality and also uniqueness of the requirements have not been highlighted. This chapter summarises the requirements from the perspective of the HARNESS platform, as described next.

	RTM (MAX)
R25	The HARNESS platform shall allow the expression of target latency or target cost
R26	The HARNESS platform shall support multiple algorithms
R27	The HARNESS platform shall exploit heterogeneous compute resources
R28	The HARNESS platform shall support fast linear read/write access to temporary data storage with understanding of duty cycle
	Delta Merge (SAP)
R29	The HARNESS platform shall provide support for specification of particular resources
R30	The HARNESS platform shall support multiple application implementations
R31	The HARNESS platform shall efficiently schedule small compute jobs
R32	The HARNESS platform shall intelligently schedule jobs to resources
R33	The HARNESS platform shall support elastic resource allocation
	adPredictor (IMP)
R34	The HARNESS platform shall support coarse-grained, resource-agnostic, goal-centric specifications of inference tasks
R35	The HARNESS platform shall efficiently schedule inference tasks on heterogeneous processors
R36	The HARNESS platform shall support in-network aggregation
R37	The HARNESS platform shall support multiple data-storage access patterns

Requirements R25 and R34 relate to the way the platform is sensitive to the customer of the application. When an application is started for a customer, the platform shall ensure the particular customer conditions can be met. These conditions can be expressed in a number of ways, including the time for application execution, cost of execution and accuracy of results.

Requirement R26 is for the platform to support applications that have multiple algorithms and the platform shall choose an appropriate algorithm. R27 and R29 require that the platform shall provision computational resources for the execution of the particular algorithm and that the required type of resources ensure execution shall meet the customers conditions. R28 and R37 are requirements that the platform shall provision a type of storage device such that an algorithm's execution can meet the customers conditions.

Requirements R30 and R35 relate to the platform supporting multiple implementations of various computation tasks that can be executed on either CPUs or accelerators, where available. R31, R32 and R35 impose requirements on the way the platform makes decisions on which implementation and resource to use. They require that a decision shall be made in a timely manner and that the result of such a decision will have a positive affect on the execution of the application. Requirement R32 states that the platform should be flexible to the changing demands of an application and, hence, have elasticity in the assignment of resources.

Requirement R36 is specific to the ability of the platform to provision in-network computation. If limitations of the network capacity for certain topologies can be overcome by using computation at specific places in the network, then the application should be able to request such a configuration. The platform shall be able to respond by using the computational facilities of in-network resources.

6 Conclusions

This report presents the three chosen industrial use cases of the HARNESS project. The HARNESS platform will develop an enhanced cloud platform upon which these use cases can be executed.

Each use case is described in detail, the logic of each application is explained and the suitability of the HARNESS platform presented. All three use cases benefit from the ability to utilise heterogeneous resources.

In order for the use cases to exploit HARNESS, a number of requirements have been identified (Sections 2.6, 3.6 and 4.6). These requirements are specific to the use cases, but they epitomise the requirements set out in the HARNESS General Requirements Report [8]. The use cases will provide a clear way to illustrate how the platform meets its requirements.

Finally, we have summarised the use-case requirements from a platform perspective (Chapter 5). There are common infrastructure required to satisfy the use cases, but also unique and innovative requirements that can showcase the HARNESS platform.

Bibliography

[1] Microsoft Bing. Available at `http://www.bing.com`.

[2] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, 2006.

[3] P. Costa, M. Migliavacca, P. Pietzuch, and A. L. Wolf. NaaS: Network-as-a-service in the cloud. In *Proceedings of the 2nd USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, Apr. 2012.

[4] A. S. Das, M. Datar, A. Garg, and S. Rajaram. Google news personalization: Scalable online collaborative filtering. In *Proceedings of the International Conference on the World Wide Web*, 2007.

[5] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters.

[6] J. Dean and S. Ghemawat. MapReduce: A flexible data processing tool. *Communications of the ACM*, 53(1), 2010.

[7] M. Dobrescu, N. Egi, K. Argyraki, B.-G. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy. RouteBricks: Exploiting parallelism to scale software routers. In *Proceedings of the ACM Symposium on Operating Systems Principles*, 2009.

[8] FP7 HARNESS project consortium. D2.1: General Requirements. Technical report, 2013.

[9] J. Gonzalez, Y. Low, C. Guestrin, and D. O'Hallaron. Distributed parallel inference on large factor graphs. In *Conference on Uncertainty in Artificial Intelligence*, 2009.

[10] T. Graepel, J. Q. Candela, T. Borchert, and R. Herbrich. Web-scale Bayesian click-through rate prediction for sponsored search advertising in Microsoft's Bing search engine. In *Proceedings of the International Conference on Machine Learning*, 2010.

[11] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. VL2: A scalable and flexible data center network. In *Proceedings of the ACM SIGCOMM Conference on Data Communication*, 2009.

[12] S. Han, K. Jang, K. Park, and S. Moon. PacketShader: A GPU-accelerated software router. In *Proceedings of the ACM SIGCOMM Conference on Data Communication*, 2010.

[13] Infer.NET. Available at `http://research.microsoft.com/en-us/um/cambridge/projects/infernet/`.

[14] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: Distributed data-parallel programs from sequential building blocks. In *Proceedings of the ACM European Conference on Computer Systems*, 2007.

[15] M. D. Linderman, R. Bruggner, V. Athalye, T. H. Meng, N. Bani Asadi, and G. P. Nolan. High-throughput Bayesian network learning using heterogeneous multicore computers. In *Proceedings of the International Conference on Supercomputing*, 2010.

[16] M. D. Linderman, J. D. Collins, H. Wang, and T. H. Meng. Merge: A programming model for heterogeneous multi-core systems. In *Proceedings of the ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2008.

[17] Mendeley. Available at `http://www.mendeley.com`.

[18] J. Naous, G. Gibb, S. Bolouki, and N. McKeown. NetFPGA: Reusable router architecture for experimental research. In *Processing of the International Workshop on Programmable Routers for Extensible Services of Tomorrow*, 2008.

[19] D. H. Stern, R. Herbrich, and T. Graepel. Matchbox: Large scale online Bayesian recommendations. In *Proceedings of the International Conference on the World Wide Web*, 2009.

[20] J. Urquhart. James hamilton, amazon vice president, on cloud economies of scale. Available at `http://news.cnet.com/8301-19413_3-20003591-240.html`.

[21] T. Willhalm, N. Popovici, Y. Boshmaf, H. Plattner, A. Zeier, and J. Schaffner. SIMD-scan: Ultra fast in-memory table scan using on-chip vector processing units. *Proceedings of the Conference on Very Large Data Bases*, 2(1):385–394, Aug. 2009.

[22] Y. Yu, P. K. Gunda, and M. Isard. Distributed aggregation for data-parallel computing: Interfaces and implementations. In *Proceedings of the ACM Symposium on Operating Systems Principles*, 2009.