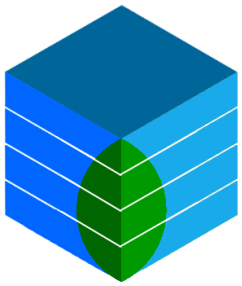


BigStorage

BigStorage: MSCA-ITN-2014-ETN-642963

Storage-based convergence between HPC and Cloud to handle Big Data

Deliverable number	D1.2
Deliverable title	Benchmarks defined & tested
Main Authors	All ESRs, A. Bilas, A. Brinkmann, A. Costan, M. Khun, A. Lebre, M. Marazakis, T. Cortes

Grant Agreement number	642963
Project ref. no	MSCA-ITN-2014-ETN-642963
Project acronym	BigStorage
Project full name	BigStorage: Storage-based convergence between HPC and Cloud to handle Big Data
Starting date (dur.)	1/1/2015 (48 months)
Ending date	31/12/2018
Project website	http://www.bigstorage-project.eu

Coordinator	María S. Pérez

Address	Campus de Montegancedo sn. 28660 Boadilla del Monte, Madrid, Spain
Reply to	mperez@fi.upm.es
Phone	+34-91-336-7380

Document Identifier	D1.2
Class Deliverable	Document
Version	1.0
Document due date	M36
Submitted	23/12/2017
Responsible	Toni Cortes, BSC
Reply to	toni.cortes@bsc.es
Document status	final
Nature	R(Report)
Dissemination level	(Public)
WP/Task responsible(s)	Toni Cortes - BSC
Contributors	All ESRs, A. Bilas, A. Brinkmann, A. Costan, M. Khun, A. Lebre, M. Marazakis, T. Cortes
Distribution List	Consortium Partners
Reviewers	All advisors
Document Location	http://bigstorage-project.eu/index.php/deliverables

Executive Summary

This deliverable presents a set of benchmarks that represent the four use cases: The Human Brain project, the Climate modelling, the Square Kilometre Array, and smart cities. These benchmarks are defined in enough detail for ESRs to use them to test their progress.

For each initiative and benchmark, we have detailed why it is important (including the requirements they fulfil), how to use and parametrize the benchmark, as well as some initial results that can help to understand the progress of ESRs in the near future.

In order to model the different use cases, we have defined (or adopted) 1 benchmark for the human brain project, 8 benchmarks for square Kilometre array, 5 benchmarks for climate, and 9 benchmarks for smart cities. With these 23 benchmarks, we cover most for the relevant requirements defined in D1.1.

Document Information

IST Project Number	MSCA-ITN-2014-ETN-642963	Acronym	BigStorage
Full Title	BigStorage: Storage-based convergence between HPC and Cloud to handle Big Data		
Project URL	http://www.bigstorage-project.eu		
Document URL	http://bigstorage-project.eu/index.php/deliverables		
EU Project Officer	Szymon Sroda		

Deliverable	Number	D1.12	Title	Benchmarks defined & tested
Workpackage	Number	WP1	Title	Use Case Analysis and Evaluation

Date of Delivery	Contractual	M36	Actual	23/12/2017
Status	version 1		final ■	
Nature	prototype □ report ■ dissemination □			
Dissemination level	public ■ consortium □			

Authors (Partner)	<name and institution>			
Responsible Author	Name	Toni Cortes	E-mail	Toni.cortes@bsc.es
	Partner	BSC	Phone	+34 934137966

Abstract (for dissemination)	In order to prove the benefits of the work done by the different proposal within BigStorage, we intend to test them (or at least most of them) in scenarios that have been detected by the European research agenda as key for Europe. Within these initiatives, we include the Human Brain project, the Climate modelling, the Square Kilometre Array, and smart cities. In this

	deliverable, we select and develop some benchmarks that represent some of the requirements of the afore mentioned use cases.
Keywords	Benchmarking, Human brain project, Square kilometre array, Climate, smart cities.

Version	Modification(s)	Date	Author(s)
01	First organization proposal	<04/10/2017>	Toni Cortes
02	First draft with all WG	<13/11/2017>	All
03	Revised draft	<18/12/2017>	All
04	Final version integrated	<19/12/2017>	Toni Cortes
05	Added comments by Reviewers	<22/12/2017>	Toni Cortes

Project Consortium Information

Participants		Contact
Universidad Politécnica de Madrid (UPM), Spain		María S. Pérez Email: mperez@fi.upm.es
Barcelona Supercomputing Center (BSC), Spain		Toni Cortes Email: toni.cortes@bsc.es
Johannes Gutenberg University (JGU) Mainz, Germany		André Brinkmann Email: brinkman@uni-mainz.de
Inria, France		Gabriel Antoniu Email: gabriel.antoniu@inria.fr Adrian Lebre Email: adrien.lebre@inria.fr
Foundation for Research and Technology - Hellas (FORTH), Greece		Angelos Bilas Email: bilas@ics.forth.gr
Seagate, UK		Sai Narasimhamurthy Email: sai.narasimhamurthy@seagate.com

DKRZ, Germany		Thomas Ludwig Email: ludwig@dkrz.de
CA Technologies Development Spain (CA), Spain		Victor Muntes Email: Victor.Muntes@ca.com
CEA, France		Jacque Charles Lafoucriere Email: Charles.LAFOUCRIERE@CEA.FR
Fujitsu Technology Solutions GMBH, Germany		Sepp Stieger Email: sepp.stieger@ts.fujitsu.com

Table of Contents

Introduction

Currently, the European research agenda includes a few flagship initiatives that have been defined as vital for Europe´s research leadership. Within these initiatives, there are four that have been identified as potential targets for the BigStorage ITN: The Human Brain project, the Climate modelling, the Square Kilometre Array, and smart cities.

These initiatives were studied during the first part of the ITN and a detailed set of requirements was defined and reported in D1.1. During the last year, we have searched for benchmarks belonging to the initiatives themselves or belonging to their areas of research to be able to test the defined requirements. In addition, when such benchmarks did not exist, we have created them by parametrizing existing generic ones in order to fill the detected gaps.

The objective of this deliverable is to present these benchmarks in enough detail for ESRs to use them to test their progress.

For each initiative and benchmark, we have detailed why it is important (including the requirements they fulfil), how to use and parametrize the benchmark, as well as some initial results that can help to understand the progress of ESRs in the near future.

The rest of the deliverable is organized into four sections, one per flagship and inside each section, all defined benchmarks are described in detail.

Use cases

Human Brain Project

Description

The Human Brain Project (HBP) is a European Commission Future and Emerging Technologies Flagship Project. It aims to put in place a cutting-edge, ICT-based scientific research infrastructure, which will allow scientific and industrial researchers to advance knowledge in the fields of neuroscience, computing, and brain-related medicine. The Project promotes collaboration across the globe and is committed to driving forward European industry.

The Neuroscience Subprojects will extend their research in brain organization and theory to support the building of increasingly sophisticated models and simulations. In addition, related work will be done in brain-like computing and robotics, working up to replicate the whole mouse brain, while also laying the foundations for simulations of the much larger and more complex human brain. The resulting knowledge on the structure and connectivity of the brain will open up new perspectives for the development of "neuromorphic" computing systems incorporating unique characteristics of the brain such as energy-efficiency, fault-tolerance and the ability to learn.

The HPC Platform Subproject (SP7) is one of the twelve operational subprojects. Its mission is to build and manage the hardware and software for the supercomputing and data infrastructure required to run cellular brain model simulations, up to the size of a full human brain. SP7 will make this infrastructure available to the consortium and the scientific community worldwide. Full brain simulations are expected to require Exascale capabilities, which according to most potential suppliers' roadmaps are likely to be available in approximately 2021-22. As well as providing sufficient computing performance, the HBP supercomputer will also need to support data-intensive interactive supercomputing and large-memory footprints. This includes topics like tightly integrated visualization, analytics, simulation capabilities, efficient Europe-wide data management, dynamic resource management providing co-scheduling of heterogeneous resources and a significant enlargement of memory capacity, based on power-efficient memory technologies.

So far researchers have focused on a small fraction of the brain, i.e., per cubic millimetre of the brain, which contains approximately 10^5 neurons. However, investigations on such a small scale are less likely to be useful for the neuroscientists who tend to consider the brain as a single entity due to its interconnectivity with other neurons.

Highly scalable computing infrastructures might allow to scale up the neural network to investigate fine-grained as well as coarse-grained levels. The major challenge in a brain-scaled simulator like the NEural Simulation Test (NEST) is the limited memory. In order to understand the limitations of such a memory intensive application, we run the NEST simulator on MOGON, scaling up the neural network from a single node to tens of nodes, with the variation of the neural network size. This study will also benefit us as researchers to better design the data structures for scalable HPC applications. We conducted and analysed the result on two nodes with 64 cores

each, and collected stats like IO, latency and used a currently being develop profiler tool for process communication. We found that by scaling the neural network, IO increases from tens of KBs to tens of MBs in a time span of 2 minutes. As we have conducted the tests in a small environment, we speculate that IOs will become a bottleneck for the performance of the application when scaling further.

The NEural Simulation Test NEST

Rationale

The basic component in the human brain simulator is the neuron and the interconnects between neurons, the synapsis, form a biological neural network. A collaboration of neuroscientists and computer scientists developed the scalable brain simulator NEural Simulation Test (NEST), which aims to run human brain as a whole [Kunkel2014][NEST] and **which has become a dominant HPC application within the HBP project.**

The neuronal infrastructure in NEST is organized as a vector containing pointers per process to the synapsis to be spiked. Initially, a single neuron is spiked which further makes connectivity with another synapse, adding more neurons on the vector. After some time, the vector is pointing to the spiking synapses that are on the local node as well as on proxy nodes.

Scientific applications show different IO patterns that can be write or read dominant or write/read transient [Ippen2017]. NEST is a write dominant application that tends to generate a large amount of data which needs to be archived for later post-processing.

NEST also enables the support of the advanced parallel interface MPI I/O to run on multiple machines. The NEST benchmark gives a flexibility to simulate multiple type of neuronal networks, which produce the I/O pattern on the underlying hardware and are bursty in nature. In general, it is a highly memory intensive application, requiring an expected maximum of 0.8 billion cores to simulate a healthy human brain, which comprises of 100 billion neurons.

How to execute it

It is necessary to compile the source code available on the project website to run the benchmark. NEST has several dependencies and many of them can be disabled at compilation time. In this benchmark, the following libraries were enabled to build the simulator binaries:

- Cython/Python - Mandatory for creating simulations with Python scripts. If Python support is disabled, NEST will only support SLI scripts (the internal simulation language for NEST).
- GNU/GSL - For solving ordinary differential equations (not necessary for our benchmark)
- Readline - For enabling command line history/editing

OpenMPI and OpenMP dependencies were enabled since they are necessary for running parallel simulations on the HPC platform MOGON at the Johannes Gutenberg University Mainz.

The simulator architecture was designed to cope with parallel simulations. A parallel simulation can run on a single desktop with a multi-core processor or in a cluster environment. It employs

the concept of a "virtual process" as the unit of execution. In a NEST parallel execution, a virtual process can be an operating system thread - for running in a single multi-core workstation - or an OpenMP thread running inside an MPI process - for running in a cluster environment.

By default, the NEST installation provides a set of predefined simulation scripts for learning and documentation purposes. The script located at

```
<installation path>/share/doc/nest/examples/hpc_benchmark.sli
```

has been selected as the basis for our benchmark deliverable. This script is parametrized by a scale factor that controls the size of the neural network. The total number of neurons is obtained by the following expression:

```
Neurons = scale_factor * 11250
```

By default, a scale factor equal to one allows running a simulation of 250 milliseconds simulation time with a single virtual process in approximately two minutes. Increasing the scale factor makes it necessary to use a more powerful server or a cluster environment.

The user specifies the number of processes in the configuration file with the type of network to run on the available node. The data structure of NEST is created during the build phase, and spiking neurons are distributed across the available nodes. Each node containing a spiking neuron is assigned a global identifier (GID)[Kunkel2017].

The NEST simulator distributes the spiking neurons across MPI ranks and OpenMP threads in a round-robin fashion to have a balanced network. Furthermore, the local threads own the data structures to enable efficient access to the neurons on the local node.

In the second step, the spiking of the neurons requires updating the data structures associated with each thread. After an update is activated, the connected neurons propagate the spiking effect in a fraction of milliseconds.

The simulation round ends with a gathering point which is also the next triggering point generated by using the MPI_Allgather. The local send buffers get ready to transfer the triggering effect to the spike registers. The send buffers have the same size on all the MPI ranks and the size depends on the number of spikes per rank during the simulation cycle.

Similarly, the buffer size changes per cycle of the spiking neural network. Other than this communication updates between the processes, transmission delay will remain the major performance bottleneck in the neural network.

Further Instructions for running NEST on MOGON

Set

- NV (virtual processes) in the script the hpc_benchmark.sli
- NP (MPI processes) on the command line

and call

1. /setenv.sh
2. cd /nest/examples
3. srun --pty -N 1 -n 64 -t 60 --mem-per-cpu 16 -p nodeshort -A zdvresearch bash –i
4. time mpiexec -n 1 nest hpc_benchmark.sli
5. nano hpc_benchmark.sli

Instructions for IOPs measurement

1. fallocate -l 10GB dummy
2. Mkfs.ext4 dummy
3. Mkdir dummyfs
4. sudo mount -o dummy dummyfs
5. losetup -a (check the loop device and change accordingly the loop0 on the next step)
6. Iostat -d -x -t -k /dev/loop0 1 1000

The parameter '1' in the `Iostat` call is the sample period and the parameter '1000' the number of iterations (after how many samples to terminate).

It is furthermore possible to continue after step 3 as follows:

4. losetup -fv ./dummy (it will print a loop device, like loopX)
5. mount /dev/loopX ./dummyfs
6. Iostat -d -x -t -k /dev/loopX 1 1000 (the rest are the same)

NEST simulator parallelization (from the NEST documentation)

A virtual process (VP) is a thread living in one of NEST's MPI processes. Virtual processes are distributed round-robin onto the MPI processes and counted continuously over all processes.

The status dictionary of each node (i.e. neuron or device) contains three entries that are related to parallel computing:

- local (boolean): indicating if the node exists on the local process or not
- thread (integer): id of the local thread the node is assigned to
- vp (integer): id of the virtual process the node is assigned to

Metrics

- Execution Time per 11,250 neurons per core: The time spent to simulate 11,250 neurons per core measured in hundreds of seconds and the effect on latency by scaling up the neuronal network size.
- Write Throughput: Number of writes issued on the disk/ssd per scaling out the neuronal network measured in Megabytes per second.
- Inter-process Communication Calls: Number of bytes send/received per processor in a collective traffic is measured with an MPI profiler.

Preliminary results

Processing Time

We measured the execution time by running the NEST benchmark on the MOGON I cluster for two nodes, each equipped with 64 cores. As in Figure 1, we found the network scale factor s = 4 to scale to all available processors because, for a lower network scaling factor as the simulation time for more than 32 processes otherwise has been almost negligible. Similarly, for a high network scale size, e.g., s=20 the simulation nearly takes infinite time to process for smaller process numbers.

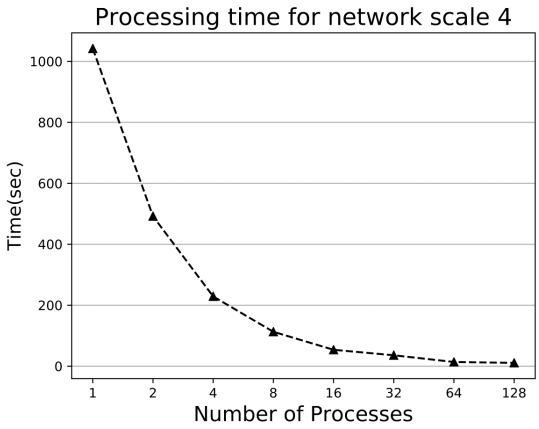


Figure 1 Execution time for a variable number of cores

Therefore, the network scale factor s=4, which contains 45,000 neurons, is the suitable case for our available 128 processors to have a clear overview of the simulation time. We observed that by increasing the number of processors exponentially, starting from 2 and reaching 128, the execution time reduces from 17.3 minutes to 11 seconds. In most of the related work, the number of processors is bound to the number of neurons, considering 11,250 neurons per processor, and then it is scaled likewise. In this case, for four processors and 45,000 neurons, MOGON executes the simulation in 250 seconds.

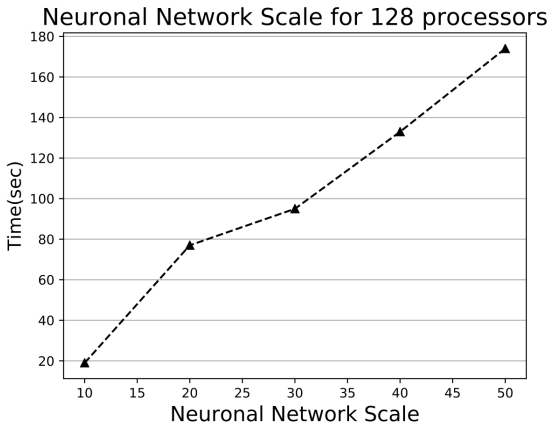


Figure 2 Execution time while scaling the neuronal network

In Figure 2, we observed a similar curve with 128 cores. We scaled up the neuronal network from 112,500 to 562,500 neurons and observed that the execution time in the beginning increase with a factor of 4 and then increase with a factor of 1.3 approximately until the maximum scale of 50.

Write Throughput

We measured the write throughput using IO stat, that helped us to determine the performance of the storage device during the benchmark. For this experiment, we selected three different network scales with two different numbers of processors, i.e 32 and 64 (see Figure 3 and Figure 4).

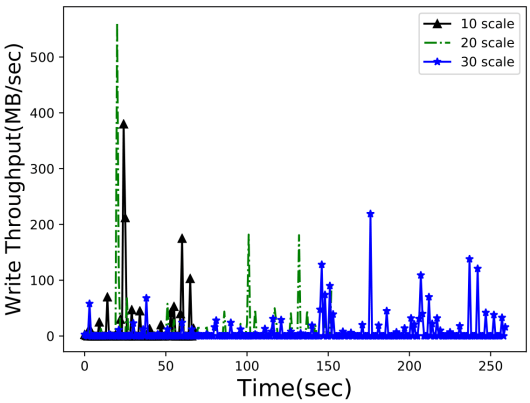


Figure 3 Write throughput for 32 cores

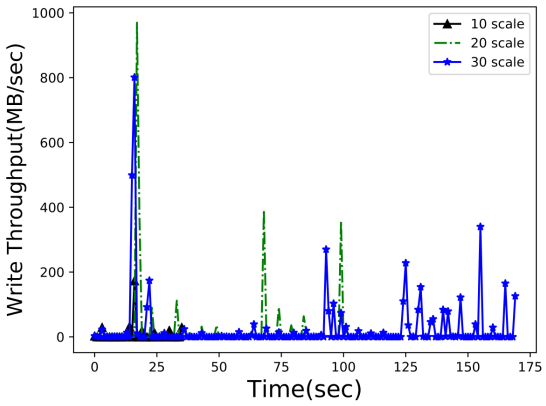


Figure 4 Write throughput for 64 cores

In Figure 5 we can see that the total time spent in the simulation for 32 processes remains relatively constant for different scale factors. On the contrary, for 64 processes the total simulation time significantly increases proportionally to the scale factor. A preliminary explanation for the increase in simulation time is deduced to be the time spent in communication time among the different processes. Given that the storage backend is used not only for data persistence but also as a communication means, we find it reasonable that that the overall data transfer time also increases. However, more experiments need to be done in order to differentiate the number of IOPS ending in the storage backend from the total amount of data being transferred. We believe that IOPS should increase proportionally to the number of processes but remain independent to the scale factor given the sequential nature of a single process.

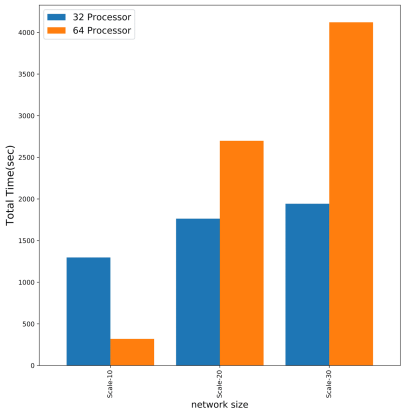


Figure 5 Total writes

Inter-process Communication Calls

In this experiment, we run the NEST simulator on a single node for six virtual processes with a neuronal network scale of two. The reason for choosing these parameters are hardware restrictions, as the used workstation only has eight cores with running at 3.6 GHz. We measured the inter-process communication by using an MPI profiler. We measured the percentage of calls

for different message sizes in bytes. For example, in Figure 6, we have defined the bin size as 20 bytes.

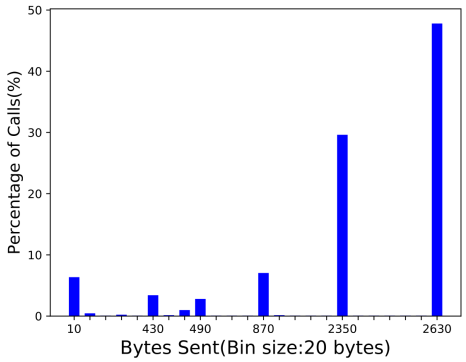


Figure 6 Collective Traffic

We found that 47.8% of the calls are in the range of message size 2,620 to 2,640 bytes and also the second largest spike with 29% calls within 2,160 to 2,180 message bytes.

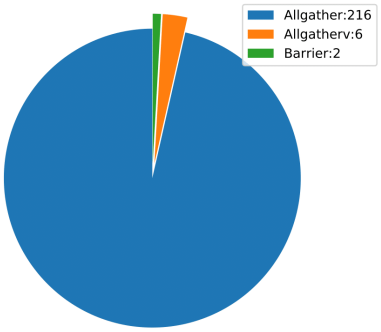


Figure 7 Calls per collective primitive

We have measured the total calls per primitive per communicator as a second experiment (seeFigure 7). Allgather dominates with 216 calls, which depicts the human brain simulator following its characteristics by connecting many-to-one and one-to-many communication patterns. As the simulator runs only for a small duration, there is a total of only six primitive calls for all_gatherv, and only two barrier calls for synchronization.

The Square Kilometre Array

Description

The Square Kilometre array is an international scientific mega-project aiming to build and operate the world's largest radio-telescope. SKA antennas will be deployed in two locations in South Africa and Australia. With a total receiving area of over a square kilometre, the goal is to surpass the capabilities of current instruments by an order of magnitude or more. The continuous stream of data that will be generated by the instrument will be processed and stored by specially designed compute centres located near the antennas' locations.

The Science Data Processor (SDP) is one of the 10 main work packages of the SKA project. The SDP encapsulates all the tasks required to design, provision and implement the necessary computing hardware, software packages and algorithms required to process the observation data into science products ready to be used by scientists. Long-term storage as well as efficient distribution of these science products are also the responsibility of the SDP.

The major challenge of SDP is handling the vast amount of data that will be generated continuously. While hardware advances are expected to catch up by the time the telescope goes live, software performance is currently at 1/1000th of the required scaling. Therefore, significant research and innovation are required to design a true exascale capable system.

Understanding the type of data to be processed by the SDP is important in order to design a suitable storage architecture. SKA data is highly noisy, large in volumes and comes from multiple observations that contain incomplete samples of the target visibility. This means that the same target will be processed through multiple iterations, creating the need for temporary storage of intermediate science products. On the other hand, the data is inherently parallelizable, allowing independent processing of partial data. This leads to the idea of "compute islands" which will be formed by partially independent compute clusters responsible for processing a subset of the incoming data stream.

It is almost certain that a multi-tiered approach will be required, combining high and low performance storage elements. Based on the expected data and computation characteristics we identify four main functions that the storage stack of the SDP will have to fulfil.

Requirements

In deliverable D1.1 we express a set of requirements (SKA1-SKA9) for managing the efficient function of the multiple storage tiers. Maximizing efficiency of data movements and optimizing for every function are the main objectives.

We have identified a set of workloads characterizing different stages, from data acquisition to archival. Each workload is characterized by a set of high-level rationales, the properties (access pattern) of those rationales, the implementation of the workload, and a preliminary evaluation.

In this deliverable and to generate the different workloads we use FIO, a versatile I/O load simulator. FIO can simulate an I/O workload given a set of parameters as input: e.g. number of

concurrent I/O, file size, I/O pattern, etc. Reproducibility is also an important reason of why we chose FIO: An experiment can easily be re-executed given the same parameters as input.

It is important to notice that there are two requirements, (SKA3, SKA7) that are not evaluated given that they are not related to performance and fall out of the scope of the work in this ITN: archiving and durability of data.

Next, we define and discuss the set of parameters that we use in our SKA benchmark.

Assumptions

Compute plane

We assume that data are generated from a single process with multiple threads, with every thread being associated to a different stream. The reason behind this assumption lies on the difference between scale-up and scale-out. When a service is composed by multiple processes, these processes can be spread over different computational nodes for load balancing and to avoid imposed bottlenecks. On the contrary, threads of a single process are subject to the same address space and I/O path and incur more dependencies. Therefore, it becomes more challenging to optimize multi-threaded processes rather than multi-process services.

Streams are generated by different sensors in the data acquisition system which can be effectively emulated by the FIO libaio engine and with a large queue depth. libaio generates concurrent I/Os that are independent from each other. Streams become correlated only at later processing stages. Note that we assume data are time-stamped during the acquisition phase prior to being stored in the system to avoid expensive OS calls at the persistence phase. A challenge in the SKA use-case is the overwhelming concurrency the storage will have to face. It is foreseen that systems will need to support billions of concurrent streams (and therefore I/Os), an unprecedented number for current storage system architectures and designs.

We assume that a process terminates when an error occurs and that cache management occurs within the scope of the application. The latter allows us to perform direct I/O. Many scientific libraries (e.g root) manage their data transparently to the application and before data land into the storage backend.

Storage plane

Before going into details about the storage plane, we first distinguish the notions of block size and file size. A block is the minimum data unit that is stored (atomically) in the backend. Given that, a file size can be characterized by the number of blocks involved.

We assume 4MB blocksize for large files and 4KB for small files. These values are not related to the access pattern (i.e. sequential or random). Note that, we assume blocks are the smallest data unit that can be accessed. Larger blocks lead to less metadata to manage. In contrast, small blocks induce more metadata to be managed, however, they allow finer grain changes to be performed.

SKAbench: Requirements and their implementation in FIO

Based on the already-defined SKA requirements, we create a benchmark that generates patters similar to the different requirements. The benchmark uses FIO with different configurations to generate each pattern. Next, we discuss the FIO parameters we use for each SKA requirement and the corresponding pattern. For each case, we also present an indicative graph with results on simple (and small) storage system configurations.

SKAbench: SKA1 Temporary storage of Raw Data

Rationale

The SDP will ingest raw data at a rate of 1 TB/s, which requires storing the raw data temporarily to perform near-real-time processing.

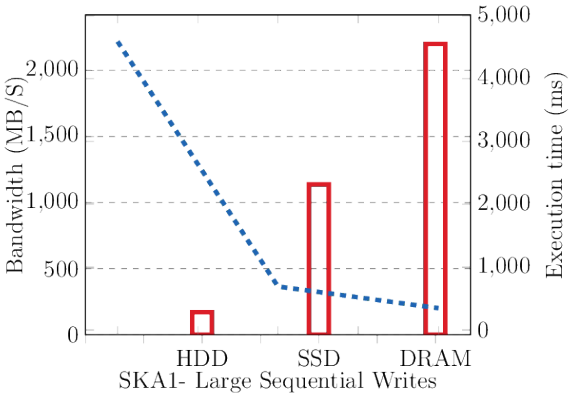
Implementation

To emulate this scenario, we assume large sequential writes in the order of Mbytes, many outstanding IOs (from 100s to 1000s), to multiple resources.

To implement the multiple resources, we have each stream resulting to a different file under the same directory.

```
BENCHMARKS
; LARGE SEQ WRITES (MB), ARGUMENT OUTSTANDING, LARGE AMOUNT OF DATA
[LARGESEQWRITE]
READWRITE=WRITE
BS=4M ;BLOCK SIZE FOR LARGE FILES
DIRECTORY=${DIRECTORY}
;NUMBER OF FILES TO USE FOR THIS JOB
NRFILES=${NUMFILES}
; RANGE FOR FILESIZE
FILESIZE=4M-${MAXFILESIZE}
IODEPTH=${OUTSTANDING}
```

Indicative Results



Example
> ./ska1.sh dir nbFiles maxFilesize(min=4MB) outstadingIOs
Example: ./ska1.sh ./testdir 1 1G 1000
dir: target mountpoint
nbFiles: number of files to be generated
maxFileSize: the maximum size of each generated file
outstandingIOs: number of concurrent I/O operations generated

Metrics

I/O Throughput, I/O operations per second (IOPS), and I/O latency.

SKAbench: SKA2 Temporary Storage of Intermediate Data Products

Rationale

Data products, in this tier, will be written once and read many times, either for further processing or to be moved to lower tiers for permanent storage.

Implementation

Create I/O request patterns that perform:

- large sequential reads (when the dataset is contiguous)
- small random reads (when the dataset is not contiguous)

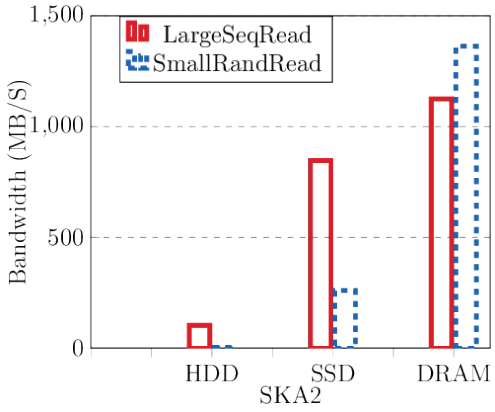
with the same parameters for the rest:

- many outstanding IOs (from 100s to 1000s)
- from a single file/device
- with a single parameter for amount of data (dataset) so we can emulate or not cache effects

Benchmark

```
[GLOBAL]
; RANGE FOR FILESIZE
FILESIZE=4M-${MAXFILESIZE}
IODEPTH=${OUTSTANDING}
DIRECTORY=${DIRECTORY}
; LARGE SEQ READS ; JOB1
[LARGESEQREAD]
READWRITE=READ
;BLOCK SIZE FOR LARGE FILES
BS=4M
;NUMBER OF FILES TO USE FOR THIS JOB
NRFILES=${NUMFILES_LSR}
```

Indicative Results



Example
>./ska2.sh dir numFiles_LargeIO numbFiles_smallRandIO maxFileSize outstadingIOs
Example: ./ska2.sh ./testdir 1 10 100m 1000
dir: target mountpoint
numFiles_LargeIO: number of files with blocksize 4MB
numFiles_smallRandIO: number of files with blocksize 4KB
maxFileSize: the maximum size of each generated file

; SMALL RANDOM READS ; JOB2 [SMALLRANDREAD] ; RANDOM READS RW=RANDREAD ;BLOCK SIZE FOR SMALL FILES BS=4K ;NUMBER OF FILES TO USE FOR THIS JOB NRFILES=${NUMFILES_SRR}	outstandingIOs: number of concurrent I/O operations generated

Metrics

I/O Throughput, I/O operations per second (IOPS), and I/O latency.

SKAbench: SKA4 Hierarchical Storage Management

Rationale

Hierarchical Storage Management solutions are currently focused mainly on archiving. A multi-tiered solution is required to efficiently manage data movement between high and low performance storage hardware. Prevision Tiers are expected to change as new technologies become available.

Implementation

- SKA4-tierup-seq/rnd
- SKA4-tierdown-seq/rnd

Create I/O request patterns that perform:

- large sequential writes to destination (to emulate write to new tier)
- many outstanding writes and reads (from 100s to 1000s)
- from a single file/device
- no parameters
- up/down means source file/device is faster/slower than destination
  1. large sequential reads from source (for contiguous datasets)
  2. small random reads from source (for non-contiguous datasets)

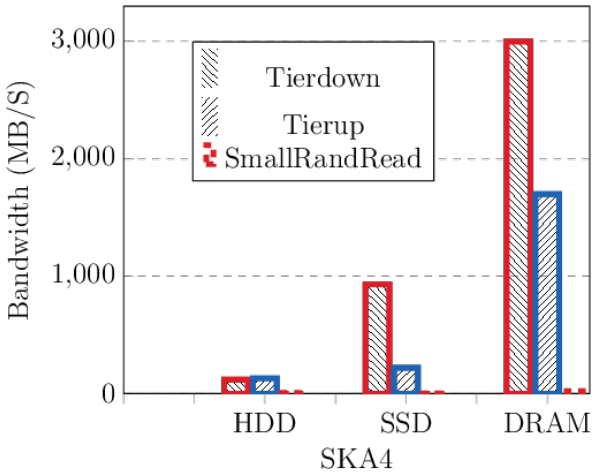
Benchmark

; LARGE SEQ WRITES (MB), ARGUMENT OUTSTANDING, LARGE AMOUNT OF DATA
[TIERDOWN]
READWRITE=WRITE
;BLOCK SIZE FOR LARGE FILES
BS=4M
DIRECTORY=${DIRECTORY}
;NUMBER OF FILES TO USE FOR THIS JOB
NRFILES=${NUMFILES}
; RANGE FOR FILESIZE
FILESIZE=4M-${LARGEFILESIZE}
IODEPTH=${OUTSTANDING}
; TIERDOWN BOTTLENECK IS THE TIER1 WRITE RATE=,${LOWERTIERWRITERATE}

[TIERUPSEQ]
READWRITE=READ
BS=4M ;THE SAME AS THE WRITTEN
DIRECTORY=${DIRECTORY}
;NUMBER OF FILES TO USE FOR THIS JOB
NRFILES=${NUMFILES}
; RANGE FOR FILESIZE
FILESIZE=4M-${LARGEFILESIZE}
IODEPTH=${OUTSTANDING}
; TIERUP BOTTLENECK IS THE TIER1 READ RATE=${LOWERTIERREADRATE},

[SMALLRANDREAD]
READWRITE=RANDREAD
;SHOULD BE SMALLER THAN WHAT IS WRITTEN
BS=4K
DIRECTORY=${DIRECTORY}
;NUMBER OF FILES TO USE FOR THIS JOB
NRFILES=${NUMFILES}
; RANGE FOR FILESIZE
FILESIZE=4K-${SMALLFILESIZE}
IODEPTH=${OUTSTANDING}
; TIERUP BOTTLENECK IS THE TIER1 READ RATE=${LOWERTIERREADRATE},

Indicative Results



Example
>./ska4.sh dir nbFiles largeFileSize smallFileSize outstandingIOs lowTierRate(Write) lowTierRate(Read)
Example: ./ska4.sh ./testdir 30 100M 10M 1 200m 10m
dir: target mountpoint
nbFiles: number of files to be generated
largeFileSize: the maximum size for files of 4MB
smallFileSize: the maximum size for files of 4KB
outstandingIOs: number of concurrent I/O operations generated
lowTierRate(Write): the emulated writing rate of low tier
lowTierRate(Read): the emulated reading rate of low tier

Metrics

I/O Throughput, I/O operations per second (IOPS), and I/O latency.

SKAbench: SKA5 Storing raw data in the processing nodes (Data Locality)

Rationale

Read and write patterns as well as locality are fairly predictable. It is therefore possible to move the storage elements very close to the actual processing, greatly reducing costly data movements.

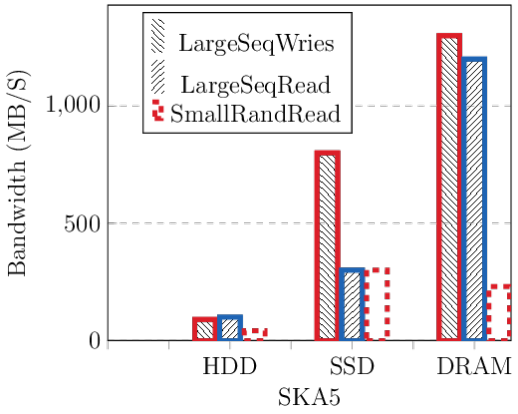
Implementation

Create I/O request patterns that emulate writing first and then reading a dataset in a single storage/address space:

- phase 1: large seq writes, many outstanding, large amount of data
- phase 2: large seq reads or small random reads, many outstanding, large amount of data

Benchmark
; large seq writes, many outstanding, large amount of data
[largeSeqWrite]
readwrite=write
;Block size for large files
bs=4M
directory=${DIRECTORY}
;Number of files to use for this job
nrfiles=${NUMFILES}
; Range for filesize
filesize=4M-${MAXFILESIZE}
iodepth=256

Indicative Results



Example
>./ska5.sh dir nbFiles sizeRangeMin sizeRangeMax
Example: ./ska5.sh ./testdir 1 4M 256M
dir: target mountpoint
nbFiles: number of files to be generated
sizeRangeMin: minimum size for the generated files
sizeRangeMax: maximum size for the generated files

Metrics

I/O Throughput, I/O operations per second (IOPS), and I/O latency.

SKAbench: SKA6 Temporary Storage of Key Data/Program State

Rationale

This data will be used for failure recovery hence it will exhibit a write often / read once pattern. Data will be duplicated via a double buffer system that allows for persistence.

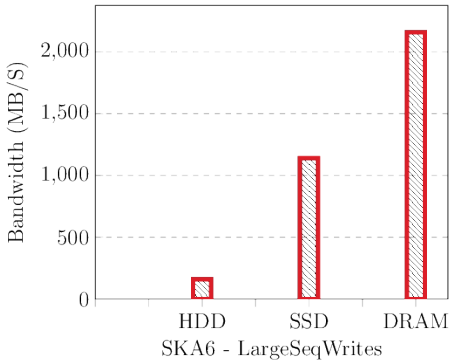
Implementation

Create I/O request patterns that emulates writing memory state to persistent storage:

- large sequential writes
- many outstanding IOs
- single file/device

Benchmark
[largeSeqWrite]
readwrite=write
bs=4M
iodepth=256
filename=${FILENAME}
size=${FILESIZE}

Indicative Results



Example
./ska6.sh target size
target: file or block device to write
size: amount of data to be written
> ./ska6.sh ./testdir/file 1G

Metrics

I/O Throughput, I/O operations per second (IOPS), and I/O latency.

SKAbench: SKA8 Read Latency Management

Rationale

This test evaluates the ability to perform explicit prefetching needed. The prefetching specification can be either provided manually via API extensions or generated automatically, e.g. generated using static code analysis.

Implementation

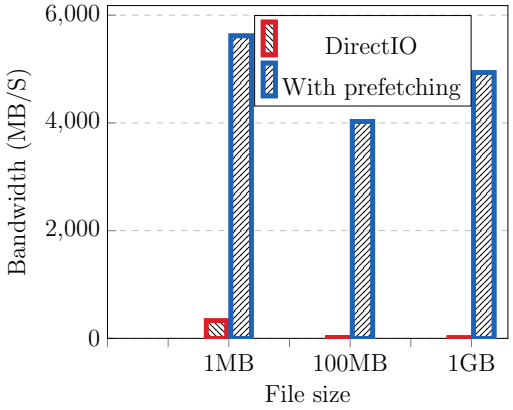
- varying strided IO reads
- With/without prefetching

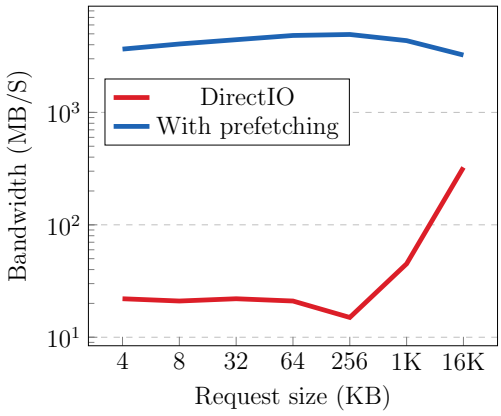
- single file/device

Benchmark

Use IOzone to generate strided accesses and examine if a system supports effective prefetching mechanisms.
./iozone -a –size=1M -i 0 -i 5
size corresponds to the file size
i corresponds to the test number and i=5 signifies strided reads.
The option –I may be used to force direct I/O to the devices.

Indicative results





Example
./iozone -a –size=1M -i 0 -i 5

Metrics

I/O Throughput, I/O operations per second (IOPS), and I/O latency.

SKAbench: SKA9 Concurrency

Rationale

This test evaluates system behaviour with respect to I/O at large degrees of concurrency. Today's data-structures do not fully exploit multi-core machine parallelism and there are opportunities to identify suitable solutions e.g., lock-free stores for high-read throughput.

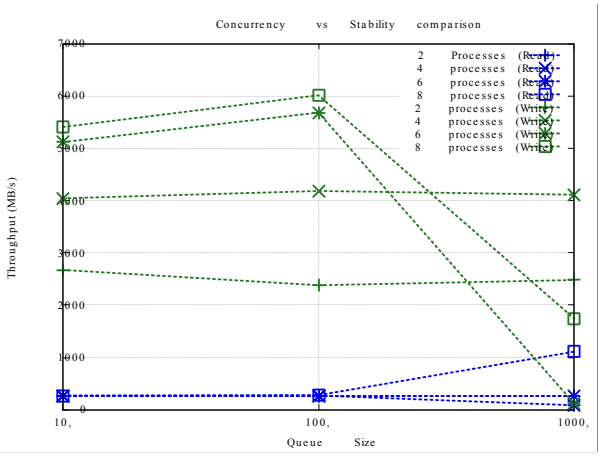
Implementation

The goal of this test is to measure throughput and scability as the amount of concurrency increases. As degree of concurrency factor we use the number of outstanding requests generated by different processes.

```
BENCHMARKS
; LARGE SEQ WRITES (MB), ARGUMENT OUTSTANDING, LARGE AMOUNT OF DATA
[LARGESEQWRITE]
READWRITE=WRITE
BS=4M ;BLOCK SIZE FOR LARGE FILES
DIRECTORY=${DIRECTORY}
;NUMBER OF FILES TO USE FOR THIS JOB
NRFILES=${NUMFILES}
; RANGE FOR FILESIZE
FILESIZE=4M-${MAXFILESIZE}
IODEPTH=${OUTSTANDING}

; LARGE SEQ READS (MB), ARGUMENT OUTSTANDING, LARGE AMOUNT OF DATA
[LARGESEQREAD]
READWRITE=READ
BS=4M ;BLOCK SIZE FOR LARGE FILES
DIRECTORY=${DIRECTORY}
;NUMBER OF FILES TO USE FOR THIS JOB
NRFILES=${NUMFILES}
; RANGE FOR FILESIZE
FILESIZE=4M-${MAXFILESIZE}
IODEPTH=${OUTSTANDING}
```

Indicative results



Example:
sudo ./ska9.sh ../ssd0/benchmarks/ 30 20M 20K 2 10

Metrics

I/O Throughput, I/O operations per second (IOPS), and I/O latency.

How to execute SKAbench

To facilitate running SKAbench, we provide a script that generates the I/O patters for each SKA requirement and outputs the required metrics. The benchmark does not require any data as input and only needs to be configured with the storage mount points that will be used as evaluation targets.

1) Install fio (e.g yum install fio)
2) Set mountpoint ./testdir, e.g mount /dev/sdX ./testdir
3) Run SKAbench ./testdir ./logdir

SKAbench is a wrapper script that handles execution of the individual benchmarks for each SKA scenario. It embeds a set of default values for the SKA benchmarks and requires only the target directory/device that needs to be measured and a directory where the execution logs will be persisted. The log for every benchmark is the output of the corresponding FIO instance.

Output

The output of all scripts is the output of the executed FIO instance, as follows:

largeSeqWrite: (g=0): rw=write, bs=(R) 4096KiB-4096KiB, (W) 4096KiB-4096KiB, (T) 4096KiB-4096KiB, ioengine=libaio, iodepth=256

fio-3.1-51-g447b

Starting 1 process

largeSeqWrite: (groupid=0, jobs=1): err= 0: pid=21269: Wed Oct 18 08:14:09 2017

write: IOPS=272, BW=1092MiB/s (1145MB/s)(1024MiB/938msec)

cpu: usr=8.11%, sys=12.38%, ctx=8450, majf=0, minf=11

IO depth: 1=0.4%, 2=0.8%, 4=1.6%, 8=3.1%, 16=6.2%, 32=12.5%, >=64=75.4%

submit: 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%

complete: 0=0.0%, 4=50.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=50.0%

issued rwt: total=0,256,0, short=0,0,0, dropped=0,0,0

latency: target=0, window=0, percentile=100.00%, depth=256

Run status group 0 (all jobs):

WRITE: bw=1092MiB/s (1145MB/s), 1092MiB/s-1092MiB/s (1145MB/s-1145MB/s), io=1024MiB (1074MB), run=938-938msec

Future Work

Our next step is to create a version of SKABench that can run on multiple nodes, in a distributed/parallel manner for evaluating a storage system from multiple mount points.

Climate Science

Description

Climate Science is the study of climate and is part of the Earth system sciences. Related topics include weather forecasting, disaster prediction and climate model research. It considers and studies the origin, development and changes of weather patterns over a long period of time. This study aims to investigate past climate, forecast future climate conditions and to see what influence it has on various aspects of our life on Earth.

Climate change has a known impact on different components of Earth system and substantial effect on human health in particular [CCS]. It drastically affects on water resources, ecosystems, food production and agriculture, industry, settlement and society, urban life, energy consumption and many other important things of our habitat and planet in whole [SGC]. The availability of information about weather and climate conditions (temperature, winds, rains, humidity, snow, thunderstorms etc.) on a daily basis became very significant for people in analysis, planning and decision making.

To predict future changes and their consequences, earth system scientists develop and maintain climate models with high-end simulations for long periods of time. These useful research tools help to better understand present and past observed climates, support experiments under given boundary conditions of anthropogenic and natural forces. Unlike these models, similar numerical weather prediction models are focused on having current meteo-information about atmospheric conditions and forecasting the future state of weather on a day-to-day basis through simulations for short periods of time.

Standard Earth system models have a comprehensive structure that consists of many components which are responsible for simulation and treatment of major parts in Earth's system. Among them usually are models of physical components (atmosphere, oceans, land-surface) and biogeochemical subsystems (aerosols, atmospheric chemistry, vegetation) [Heavens 2013]. Execution of such complex applications with numerical models of the climate system demands powerful computing infrastructure that will support it. Thus, climate science applications are key users of HPC which still continue to improve and drive scientific progress.

Unfortunately, the progress is limited by computer capabilities. Even though today's computing power of supercomputers is very advanced and still increases, the complexity of climate models also increases and requires more computer resources than before or even more than affordable. For example, large scale experiments with help of climate models require longer simulation runs to forecast climate change during a century which means using more processor hours. Among other main problems in HPC while working with climate models is handling of huge amounts of data produced by simulations. Climate models with high resolution would not produce valuable results as they are constrained by available resources: storage resources, proper data management and post-processing.

Currently, more and more climate models are being developed and improved by earth system scientists and computational experts together with large computing centres. It is important to work closely with climate scientists to improve scientific applications and find the best computational methods and algorithms, programming languages and techniques. All this will foster the improvement of current supercomputer systems and enhance the development of Earth system models.

In D1.1 we have identified multiple enabling requirements for climate science data storage and processing, which were divided into seven categories:

- **I/O requirements** enabling high-performance, parallel access to the data [CLM1, CLM2]
- **Data management requirements** necessary to ensure that both the large volumes of source and result data remain accessible indefinitely for reprocessing or further processing [CLM3, CLM5]
- **Data processing requirements** necessary to ensure the need for high spatial and temporal resolution in order to improve the fidelity of climate models or the predicted new era of climate forecasting [CLM4]
- **Data standards and interoperability requirements** guaranteeing that the stored data remains usable and easily linkable to new experiments if necessary, thanks to well-defined data/metadata standards and data quality assessment [CLM6, CLM7]
- **Climate data acquisition to data access requirements** to ensure that meaningful data can be used and shared efficiently amongst scientists and institutions around the world [CLM8]
- **Software infrastructure requirements** allowing the source code used to perform experiments to remain accessible, usable, and maintainable and over time [CLM9]
- **Social requirements** to foster trans-domain collaboration between climate scientists and software engineers [CLM10]

There is a variety of profiling or tracing tools and benchmarks available currently for scientific community which are useful for performance evaluation of different computing systems. However, in this section we selected for our purposes only those, which can be leveraged by scientists and developers who aim to improve infrastructure and software used in Climate Science. Below we describe in more detail every chosen benchmark and tool, explaining why it was considered, which Climate use case requirements it satisfies, what it measures, how it can be executed and used by researchers.

IOR

IOR is designed to measure parallel file system I/O performance at both the POSIX and MPI-IO level. "IOR" stands for "Interleaved Or Random," which has very little to do with how the program works currently. It focuses on measuring the sequential read/write performance under different file size, I/O transaction size, and concurrency. It also differentiates the strategies to use a shared file or one file per processor. More importantly, it supports both the traditional POSIX interface and the advanced parallel I/O interfaces, including MPI I/O, HDF5, HDFS, S3, S3_EMC, or NCMPI.

These alternative file strategies can be directly compared head-to-head for an identical set of testing parameters. Many US laboratories are having their own flavour [IOR-LANL, IOR-LLNL].

Rationale

This benchmark is in fact a standard tool to measure the performance of parallel file systems which are an important component of HPC infrastructure needed for climate applications. IOR can help ESRs to imitate the output of a climate science application in order to be tested in all the variety of I/O interfaces and access patterns. This can help researchers to choose which underlying interface will boost their application (POSIX or MPI-IO for instance). The output can be captured for storage. Also, it outputs the command for better reproduction of the results. IOR is a low-level benchmark so it can also measure differences between underlying media. To sum up, this benchmark completely satisfies CLM1 and CLM2 requirements described in D1.1.

Metrics

IOR benchmark allows user to collect next metrics:

- *Max(MiB)* - the maximum amount in mebibyte for that operation
- *Min(MiB)* - the minimum amount in mebibyte for that operation
- *Mean(MiB)* - the mean amount in mebibyte for that operation
- *StdDev* - the standard deviation in mebibyte for that operation
- *Mean(s)* - the mean amount of total seconds for that operation

How to execute it

The main component that IOR needs is the openmpi library (libopenmpi-dev) because it needs mpicc and files that are associated with this library. The user can get the latest source code form the various git repositories [IOR-LANL, IOR-LLNL].

```
user@hostname:~$ git clone https://github.com/LLNL/ior.git
Cloning into 'ior'...
remote: Counting objects: 535, done.
remote: Total 535 (delta 0), reused 0 (delta 0), pack-reused 535
Receiving objects: 100% (535/535), 289.96 KiB | 0 bytes/s, done.
Resolving deltas: 100% (326/326), done.
Checking connectivity... done.
        user@hostname:~$ cd ior
        user@hostname:~/ior$ ./bootstrap
        user@hostname:~/ior$ ./configure --prefix="<download directory>"
        user@hostname:~/ior$ make install
        user@hostname:~/ior$ cd ../bin/
```

If interested user leverages Grid'5000 supercomputers [g5k] to deploy and use IOR benchmark then it is necessary to have privileges for installing in */usr/local*.

- Below is presented the list of some parameters. The full list can be found in [IOR-FLAGS]

  -a S api -- API for I/O [POSIX|MPIIO|HDF5|NCMPI]

-d N interTestDelay -- delay between reps in seconds
-e   fsync -- perform fsync upon POSIX write close
-g   intraTestBarriers -- use barriers between open,write/read, and close
-s N segmentCount -- number of segments
-v   verbose -- output information (repeating flag increases level)
-F   filePerProc -- file-per-process (single or multiple files)
-C   reorderTasks -- changes task ordering to n+1 ordering for readback
-B   useO_DIRECT -- uses O_DIRECT for POSIX, bypassing I/O buffers
-k   keepFile -- don't remove the test file(s) on program exit
-x   singleXferAttempt -- do not retry transfer if incomplete
-w   writeFile -- write file
-r   readFile -- read existing file
-t N transferSize -- size of transfer in bytes (e.g.: 8, 4k, 2m, 1g)

- An example of simple Run is shown below:

```
user@hostname:~/bin$ ./ior -a POSIX
--------------------------------------------------------------------------
[[5057,1],0]: A high-performance Open MPI point-to-point messaging module
was unable to find any relevant network interfaces:

Module: OpenFabrics (openib)
  Host: hostname

Another transport will be used instead, although this may result in
lower performance.
--------------------------------------------------------------------------
IOR-3.0.1: MPI Coordinated Test of Parallel I/O

Began: <Date> <Time>
Command line used: ./ior -a POSIX
Machine: Linux hostname

Test 0 started: <Date> <Time>
Summary:
api                = POSIX
test filename      = testFile
access             = single-shared-file
ordering in a file = sequential offsets
ordering inter file= no tasks offsets

clients            = 1 (1 per node)
repetitions        = 1
xfersize           = 262144 bytes
blocksize          = 1 MiB
aggregate filesize = 1 MiB

access    bw(MiB/s)  block(KiB) xfer(KiB)  open(s)    wr/rd(s)   close(s)   total(s)   iter
------    ---------  ---------- ---------  --------   --------   --------   --------   ----
write     185.33     1024.00    256.00     0.000289   0.001827   0.003276   0.005396   0
read      662.19     1024.00    256.00     0.000451   0.001041   0.000015   0.001510   0
remove    -          -          -          -          -          -          0.000526   0

Max Write: 185.33 MiB/sec (194.33 MB/sec)
Max Read:  662.19 MiB/sec (694.36 MB/sec)

Summary of all tests:
Operation   Max(MiB)   Min(MiB)  Mean(MiB)     StdDev    Mean(s) Test# #Tasks tPN reps fPP reord reordoff reordrand seed segcnt
blksiz xsize aggsize API RefNum
write       185.33     185.33     185.33       0.00    0.00540 0 1 1 1 0 0 1 0 0 1 1048576 262144 1048576 POSIX 0
```

read 662.19 662.19 662.19 0.00 0.00151 0 1 1 1 0 0 1 0 0 1 1048576 262144 1048576 POSIX 0

Finished: <Date> <Time>

- Only read example

user@hostname:~/bin$ ./ior -a POSIX -N 1 -d 5 -o /home/<user>/testior1 -e -g -r -i 4 -vv -F -C -k | tee -a outputfilename
--
[[23013,1],0]: A high-performance Open MPI point-to-point messaging module was unable to find any relevant network interfaces:

Module: OpenFabrics (openib)
Host: hostname

Another transport will be used instead, although this may result in lower performance.
--
IOR-3.0.1: MPI Coordinated Test of Parallel I/O

Began: <Date> <Time>
Command line used: ./ior -a POSIX -N 1 -d 5 -o /home/<user>/testior1 -e -g -r -i 4 -vv -F -C -k
Machine: Linux hostname 3.16.0-4-amd64 #1 SMP Debian 3.16.43-2+deb8u2 (2017-06-26) x86_64
Using synchronized MPI timer
Start time skew across all tasks: 0.00 sec

Test 0 started: Wed Nov 8 16:00:14 2017
Path: /home/<user>
FS: 9.9 TiB Used FS: 31.1% Inodes: 320.0 Mi Used Inodes: 4.1%
Participating tasks: 1
Using reorderTasks '-C' (expecting block, not cyclic, task assignment)
task 0 on hostname
Summary:
api = POSIX
test filename = /home/<user>/testior1
access = file-per-process
pattern = segmented (1 segment)
ordering in a file = sequential offsets
ordering inter file= constant task offsets = 1
clients = 1 (1 per node)
repetitions = 4
xfersize = 262144 bytes
blocksize = 1 MiB
aggregate filesize = 1 MiB
Commencing read performance test: Wed Nov 8 16:00:19 2017
Using Time Stamp 1510153214 (0x5a031bfe) for Data Signature

access bw(MiB/s) block(KiB) xfer(KiB) open(s) wr/rd(s) close(s) total(s) iter
------ --------- ---------- --------- -------- -------- -------- -------- ----
delaying 5 seconds . . .

```
read  547.63    1024.00   256.00  0.000868  0.000877  0.000017  0.001826  0
Commencing read performance test: Wed Nov  8 16:00:24 2017
Using Time Stamp 1510153219 (0x5a031c03) for Data Signature
delaying 5 seconds . . .
read  590.66    1024.00   256.00  0.000522  0.001073  0.000020  0.001693  1
Commencing read performance test: Wed Nov  8 16:00:29 2017
Using Time Stamp 1510153224 (0x5a031c08) for Data Signature
delaying 5 seconds . . .
read  494.79    1024.00   256.00  0.000828  0.001105  0.000021  0.002021  2
Commencing read performance test: Wed Nov  8 16:00:34 2017
Using Time Stamp 1510153229 (0x5a031c0d) for Data Signature
delaying 5 seconds . . .
read  595.61    1024.00   256.00  0.000556  0.001036  0.000021  0.001679  3


Max Read:  595.61 MiB/sec (624.55 MB/sec)


Summary of all tests:
Operation   Max(MiB)   Min(MiB)  Mean(MiB)     StdDev    Mean(s) Test# #Tasks tPN reps fPP reord reordoff reordrand
seed segcnt blksiz xsize aggsize API RefNum
read       595.61     494.79     557.17      40.57    0.00180 0 1 1 4 1 1 1 0 0 1 1048576 262144 1048576 POSIX 0


Finished: <Date> <Time>
```

NetCDF-Bench

NetCDF Performance Benchmark Tool (NetCDF-Bench) [nbench_git] is a performance benchmark tool which was developed in German Climate Computing Center in cooperation with Bull to measure NetCDF I/O performance [netcdf] on different systems and devices. This benchmark mimics the typical I/O behaviour of scientific climate applications and captures the performance on each node or process. In the end, it collects all the obtained results with related data and provides them in the human-readable format.

NetCDF-Bench supports different I/O modes, among which are independent I/O, collective I/O and chunked I/O. Benchmark can also pre-fill the variables with some value if it is necessary during the performance measurements. It provides support for various access patterns including accesses to 4D datasets (one time dimension and three data dimensions).

Rationale

Climate scientists today are using Network Common Data Form (NetCDF) file format [netcdf] to store all amounts of data produced by climate applications after the large-scale experiments. One can observe that the amount of data, and consequently, the time spent for I/O is growing constantly over time, which is an undesirable tendency. To relieve such a situation, different approaches can be used as well as optimization. A precise benchmark tool provides opportunities to see the impact of optimizations performed by developers. NetCDF-bench implementation was highly necessary because unfortunately there is no suitable benchmarks for NetCDF file format that can evaluate its I/O performance on HPC systems.

Metrics

NetCDF-Bench [nbench_vi4io] collects and evaluates minimal, average and maximal values and ratios for next metrics:

- *Open time* - time needed to open a certain file.
- *I/O time* - time needed to for writing all the data into file.
- *Close time* - time necessary for closing the opened and filled with data file.
- *I/O performance* (w/o open/close) - the speed in MB/s of writing data into an opened file without considering the time necessary for open and close operations.
- *I/O performance* - the speed in MB/s of writing data into a file where time needed for open and close operations is included.

How to execute it

Before running NetCDF-Bench it is necessary to install all crucial high-level I/O libraries. For a manual installation of required software stack, please see the list of commands below:

- Firstly, user needs to install zlib compression library [zlib] required by NetCDF and HDF5 ttp://www.zlib.net. The zlib compression library provides in memory compression and decompression functions, including integrity checks of the uncompressed data.

  root@hostname:~# wget http://zlib.net/zlib-1.2.11.tar.gz
  root@hostname:~# tar xf zlib-1.2.11.tar.gz
  root@hostname:~# cd zlib-1.2.11
  root@hostname:~/zlib-1.2.11# ./configure --prefix=/usr/local
  root@hostname:~/zlib-1.2.11# make
  root@hostname:~/zlib-1.2.11# sudo make install
- If necessary, High-Performance Portable MPI library [mpi] can be installed in this way:

  root@hostname:~# wget http://www.mpich.org/static/downloads/3.2/mpich-3.2.tar.gz
  root@hostname:~# tar xf mpich-3.2.tar.gz
  root@hostname:~# cd mpich-3.2
  root@hostname:~/mpich-3.2# ./configure --prefix=/usr/local
  root@hostname:~/mpich-3.2# make
  root@hostname:~/mpich-3.2# sudo make install
- Then user needs to install HDF5 library [hdf5]. There are two releases of HDF5 which are currently available for download. HDF5-1.10 can read files created with earlier releases, but earlier releases such as HDF5-1.8 may not be able to read HDF5-1.10 files.

  root@hostname:~# wget https://support.hdfgroup.org/ftp/HDF5/current18/src/hdf5-1.8.18.tar
  root@hostname:~# tar xf hdf5-1.8.18.tar
  root@hostname:~# cd hdf5-1.8.18
  root@hostname:~/hdf5-1.8.18# ./configure --prefix=/usr/local
  root@hostname:~/hdf5-1.8.18# make
  root@hostname:~/hdf5-1.8.18# sudo make install
- Then user needs to install NetCDF-4 by executing next commands

  root@hostname:~# wget ftp://ftp.unidata.ucar.edu/pub/netcdf/netcdf-4.5.0.tar.gz
  root@hostname:~# tar xf netcdf-4.5.0.tar.gz

```
root@hostname:~# cd netcdf-4.5.0
root@hostname:~/netcdf-4.5.0# ./configure --prefix=/usr/local
root@hostname:~/netcdf-4.5.0# make
root@hostname:~/netcdf-4.5.0# sudo make install
```

- Then user needs to install netCDF-Fortran library(or depending on users needs netCDF-C):

```
root@hostname:~# wget ftp://ftp.unidata.ucar.edu/pub/netcdf/netcdf-fortran-4.4.4.tar.gz
root@hostname:~# tar xf netcdf-fortran-4.4.4.tar.gz
root@hostname:~# cd netcdf-fortran-4.4.4/
root@hostname:~/netcdf-fortran-4.4.4# ./configure --prefix=/usr/local
root@hostname:~/netcdf-fortran-4.4.4# make
root@hostname:~/netcdf-fortran-4.4.4# sudo make install
```

- Alternatively, Spack can be used to install all dependencies describe above in this list:

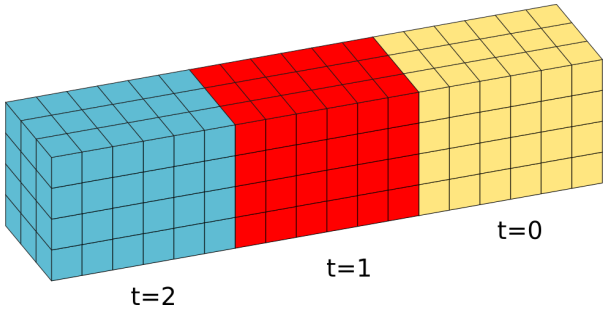
```
$ git clone https://github.com/spack/spack.git
$ cd spack
$ . ./share/spack/setup-env.sh
$ spack install zlib mpich hdf5 netcdf netcdf-fortran
$ spack load zlib mpich hdf5 netcdf netcdf-fortran
```

- Afterwards, NetCDF-Bench benchmark can be installed by downloading its source code from github [nbench_git].

NetCDF-Bench (released under GNU LGPL license) itself was designed to run without any parameters, however for advanced usage this tool provides a number of parameters (see below).

```
Benchtool (datatype: int)
Synopsis: ./benchtool [-n] [-p] [-d] [-b] [-c] [-r] [-w] [-t] [-u] [-f] [-x] [-F] [--verify]  [Optional Args]
Flags
-r, --read                Enable read benchmark
-w, --write               Enable write benchmark
-u, --unlimited           Enable unlimited time dimension
-F, --use-fill-value      Write a fill value
--verify                  Verify that the data read is correct (reads the data again)
Optional arguments
-n, --nn=0                Number of nodes
-p, --ppn=0               Number of processes
-d, --data-geometry=STRING    Data geometry `(t:x:y:z)`
-b, --block-geometry=STRING   Block geometry `(t:x:y:z)`
-c, --chunk-geometry=STRING   Chunk geometry `(t:x:y:z|auto)`
-t, --io-type=ind         Independent / Collective I/O (ind|coll)
-f, --testfile=STRING     Filename of the testfile
-x, --output-format=human     Output-Format (parser|human)
```

In order to understand what do all the optional arguments mean, it is necessary to consider Domain decomposition. On the picture below an example of data with geometry (3:6:4:3) is depicted:



The data is written in blocks to the shared file. The block size can be customized, but there are some restrictions. Assume, that

- *t:x:y:z* is the data size
- *t* is a multiple of some integer value *s*
- *nn* is the number of nodes
- *ppn* is the number of processes per node
- *px* and *py* are integer values, which satisfy the condition: $px \cdot py = nn \cdot ppn$
- Then (*s:x/px:y/py:z*) is a valid block size. (Default block size is (*1:x/nn:y/ppn:z*).)

Each process allocates (*s:x/px:y/py:z*) * *type_size* memory space and the data is read/written in timesteps.

After the execution on chosen platform, NetCDF-Bench aggregates the result of all processes and creates a summary. Here is an example of output:

```
$ mpiexec -n 1 ./benchtool
[1494497686.787267] [mlogin103:1936 :0]        sys.c:744  MXM  WARN  Conflicting CPU frequencies detected, using: 2501.00
Benchtool (datatype: int)
Data geometry (t:x:y:z x sizeof(type))     100:100:100:10 x 4 bytes
Block geometry (t:x:y:z x sizeof(type))     1:100:100:10 x 4 bytes
Datasize 40000000 bytes (40.0 MB)
Blocksize 400000 bytes (400.0 kB)
I/O Access independent
Storage contiguous
File length fixed
File value no
                                        min              avg              max
Benchmark:write  Open time          0.2811507931     0.2811507931     0.2811507931 secs
Benchmark:write  I/O time           0.1901479111     0.1901479111     0.1901479111 secs
Benchmark:write  Close time         0.3576489800     0.3576489800     0.3576489800 secs
Benchmark:write  I/O (w/o o/c)      200.6173638152   200.6173638152   200.6173638152 MiB/s
Benchmark:write  I/O Performance    46.0185526612    46.0185526612    46.0185526612 MiB/s
```

MACSio

MACSio is a Multi-purpose, Application-Centric, Scalable I/O proxy application [macsio_llnl] which allows users to make testing and evaluation of parallel I/O performance for a wide variety of real-

world applications which belong to different scientific domains. Due to its design, many I/O libraries and I/O paradigms can be compared and tested in order to predict the scalability performance of certain applications in HPC or even Cloud computing.

MACSio design [macsio_design] has two key features that allow it to mimic I/O workloads and help to distinguish from existing today I/O proxy applications and benchmarking tools (see Figure 8). One of them is the level of abstraction (LOA) where the benchmark orchestrates tests and monitors performance (POSIX, MPI-IO, SILO, HDF5 and beyond). Another one is the degree of flexibility which provides an opportunity to assess a variety of I/O interfaces and parallel I/O paradigms through support of many programming APIs.

Rationale

Results obtained from MACSio benchmark usage are very helpful to see how climate applications can be improved regarding their I/O performance. This software provides evaluation of HDF5 I/O performance which is one of the most popular and usable library in climate science. In addition, MACSio provides also evaluation of compression for multi-dimensional arrays. Thus, this benchmark satisfies requirements CLM1, CLM2, CLM3 and CLM6 defined in deliverable D1.1.

Different benchmarks available today for a scientific community provide mostly performance assessment to a single I/O interface, rarely to a several libraries. The implementation and development of suchlike benchmarks for the performance evaluation need a lot of efforts and expert knowledge of the certain I/O library. However, MACSio provides support to variety of I/O interfaces because its design uses specific plugins coded for exact library [macsio_paper1]. Currently benchmark employs such interfaces like exodusII, HDF5, MPI-IO, Silo. Through such plugin-based design MACSio achieves Multi-purpose and if user wants to assess own library then only its plugin must be written and integrated.

Metrics

MACSio provides ability to test and compare various I/O libraries and I/O paradigms, to predict scalable performance of real applications and to help identify where improvements in I/O performance can be made. This tool generates a dataset dump for such purposes and collects next characteristics related to input/output of data:

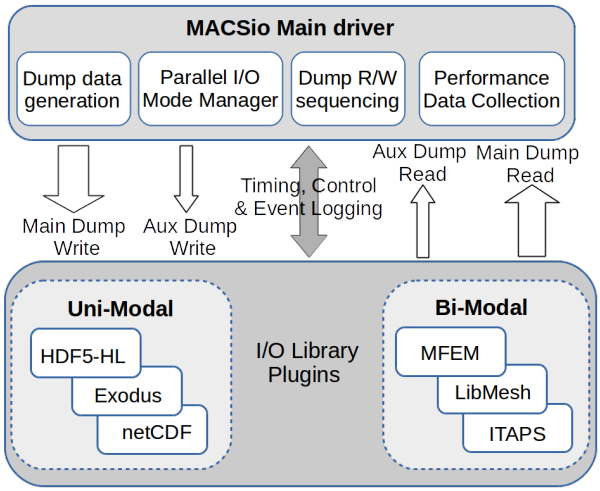


Figure 8 High Level Design of MACSio [macsio_design]

- Write time - the minimal time required by plugin to complete dump writing
- Read time - the minimal time needed for a plugin to complete dump reading
- Average time for both Read and Write for a sequence of dumps
- I/O bandwidth = dumpBytes/dumpTime
- Total, cumulative and slowest individual I/O times for various libraries

In addition, this benchmark also provides testing of compression for multi-dimensional arrays, usage and performance of which is much more complicated than swaths of 1D buffers.

How to execute it

MACSio is released under the terms of the GPL license and can be used by any scientific community. It leverages GNU Makefiles with conditionally constructed variables and shell functions. The source code of MACSio is divided into two key directories where its main functionality located in the */macsio* and all available plugins are in the */plugins* directory [macsio_doc].

At first MACSio main should be builded. The main bootstrap for building MACSio is the *config-site* file. This file contains variable definitions for all the key Make variables necessary to control the build of MACSio and any of its plugins. Example of *config-site* file can be find below:

```
SILO_HOME = /Users/<user_name>/visit/visit/silo/4.10.2-h5par/i386-apple-darwin12_gcc-4.2
HDF5_HOME = /Users/<user_name>/visit/visit/hdf5/1.8.11-par/i386-apple-darwin12_gcc-4.2
ZFP_HOME  = $(HDF5_HOME)
EXODUS_HOME = /Users/<user_name>/Downloads/exodus-6.09/exodus/myinstall
NETCDF_HOME = /Users/<user_name>/visit/thirdparty_shared/2.8/netcdf/4.3.2/i386-apple-darwin12_gcc-4.2
CXX = /Users/<user_name>/installs/openmpi/1.6.4/i386-apple-darwin12_gcc-4.2/bin/mpicxx
CC  = /Users/<user_name>/installs/openmpi/1.6.4/i386-apple-darwin12_gcc-4.2/bin/mpicc
CFLAGS = -DHAVE_MPI -g
LINK = $(CXX)
```

At the second step, MACSio Plugins must be builded. The benchmark builds automatically those that depend upon ubiquitous system libraries such as stdio. All the gathered results can be accessed by reading the benchmark output in macsio-log.log.

Interested user can find a useful information in [macsio_doc] with more detailed instructions and tips about MACSio usage. An example of output data is presented at [macsio_vi4io].

Big Brother File System interceptor (BBFS)

BBFS [bbfs_git] is a simple utility for Linux that proxies storage operations to a POSIX-compliant file system from a virtual mount point to another mount point. It does so while logging all requests and metadata that are proxied to a file. Doing so, it allows the user to detail the exact storage operations performed by any application to the virtual mount point without ever needing to modify the application code. Such proxying is completely transparent for the application, which will only need to have its working directory changed from the original mount point to the virtual mount point.

It relies on FUSE [fuse], that is available in all recent Linux kernels. It has been proven to work on various platforms, including Grid'5000, as well as the Cooley and Theta supercomputers at Argonne National Laboratory.

Rationale

The output from this tool can be used to:

- Build a benchmark tool that will replay the results without requiring the original application.
- Study the actual behavior of the application to understand its I/O patterns and behavior. This has already been used in that context in published, peer-reviewed work [bbfs_paper].
- Alternatively, it could be used for debugging an application

In general, BBFS is satisfying CLM1, CLM2, and CLM4 requirements from deliverable D1.1. It corresponds to first two requirements because of climate applications' I/O behavior evaluation on various platforms and with different I/O interfaces. The last one is satisfied because BBFS allows users to test the scalability of application and to see which storage capacity is needed for a chosen application either on HPC platform or cloud computing infrastructure.

Metrics

BBFS utility allows user to measure next metrics:

- Total reads - the size of all data which were dumped to a storage
- Total writes - the size of all data which were accessed by application
- Read/Write ratio
- Read and write times - time spend for data manipulation
- I/O throughput

How to execute it

To install interceptor, user needs git, cmake, FUSE development libraries, and a C compiler. If ant climate application is available and has a data directory on /data/myapp, then it is possible to monitor all storage calls the application makes during its lifespan.

First, user needs to get and compile bbfs.

git clone https://github.com/pierreis/bbfs
cd bbfs
cmake .
make

On another console, user should go to the bbfs folder and execute it. Say that proxied folder pointing to */data/myapp* will be on */tmp/myapp* (user is free to choose whatever is pleased as long as the proxied folder and virtual folder do not overlap), and afterwards all calls will be logged in a text file located at *~/mylogfile.txt*. Execute:

./bbfs [FUSE options] /data/myapp /tmp/myapp ~/mylogfile.txt

The execution blocks. BBFS is active. User must Run the application on the other console, setting its data directory to */tmp/myapp*.

Once the application terminates, bbfs can killed. The log file will contain all calls made by the application to the virtual folder.

Preliminary results

The chosen application in a left column of Table 1 were deployed on the Grid'5000 experimental testbed, distributed over 11 sites in France and Luxembourg. For these experiments, the parapluie cluster of Rennes was used. Each node is outfitted with 2 x 12-core 1.7 Ghz 6164 HE, 48 GB of RAM, and 250 GB HDD. Network connectivity can be handled either by Gigabit Ethernet connectivity (MTU = 1500 B) or by 4 x 20G DDR InfiniBand.

We run all applications and analyse all storage calls they perform. We log these calls for chosen applications using BBFS interceptor. The results for these HPC applications are obtained by using 24 compute / 8 storage nodes. Using 4 or 12 storage nodes does not lead to any significant difference in the results.

Table 1 Results obtained from BBFS usage for HPC applications

Application	Usage	Total reads	Total writes	R / W ratio	Profile
mpiBLAST [blast]	Protein docking	27.7 GB	12.8 MB	2.1×10^3	Read-intensive

Modular Ocean Model (MOM) [mom]	Oceanic model	19.5 GB	3.2 GB	6.01	Read-intensive
ECOHAM [ecoham]	Sediment propagation	0.4 GB	9.7 GB	4.2×10^{-2}	Write-intensive
Ray Tracing [raytracing]	Video processing	67.4 GB	71.2 GB	0.94	Balanced

More about details about these results and their comparison with applications from Cloud computing can be found in published paper [bbfs_paper].

Score-P measurement system for parallel performance

Description

Score-P [scorep] is a measurement infrastructure that supports profiling, tracing and online analysis of HPC applications. Developed under BSD 3-Clause License, it has a set of instrumentation tools which allow users to integrate measurement scripts into their C/C++ and Fortran codes. In this way the performance-related data regarding times, visits, communication metrics, or hardware counters can be gathered during application runs. All the traces obtained after measurements can be used afterwards to replay application I/O.

By default, Score-P runs in profiling mode and produces data in the CUBE4 [cube] format. This data gives an insight about bottlenecks in efficiency of application and allows to customize options for subsequent measurement runs, for example, to select MPI groups to record, to specify the total amount of memory the measurement is allowed to consume, to adapt trace output options, to specify a set of hardware counters to be recorded, and many others [scorep_chapter].

On Figure 9 below is presented a general overview of the Score-P infrastructure which includes instrumentation framework, several runtime libraries, measurement and the analysing tools. At the lowest level it supports programming models and other event sources and provides instrumentation tools with the necessary code to the application at build time. Score-P stores all the performance data in the open data formats CUBE4 for call-path profiles and OTF2 for event traces [toolsguide]. This provides opportunity for a multiple analysis tools to use the same data.

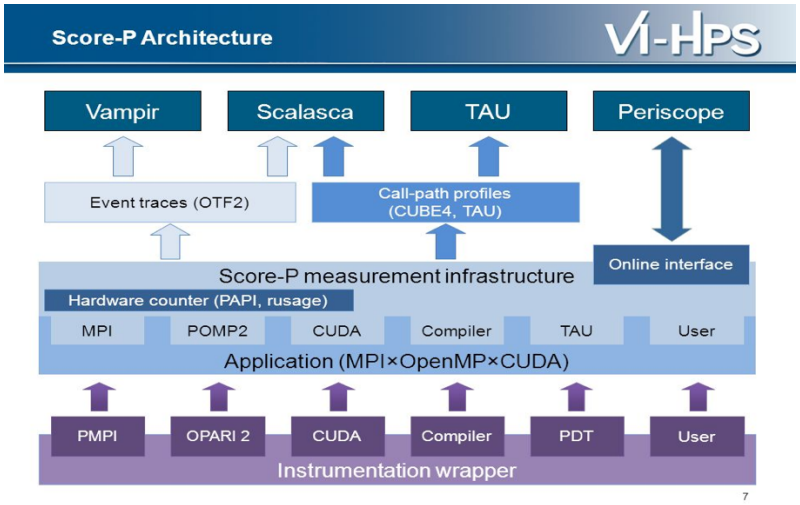


Figure 9 General scheme of Score-P instrumentation and measurement infrastructure including produced dataformats and analysing tools [scorep_prs].

Rationale

Score-P supports next parallel paradigms which are often used in HPC and response to requirement CLM1 described in Deliverable D1.1:

- Multi-process:MPI, SHMEM
- Thread-parallel:OpenMP, Pthreads
- Accelerator-based:CUDA, OpenCL, OpenACC
- This framework provides climate application users a single platform with infrastructure for a large-scale performance measurements. This framework is packed with variety of analysis tools where Open Trace Format Version 2 (OTF2) [otf2] is the important one for Climate use case.

The OTF2 is a designed software package which performs I/O recording and stores events of interest (like enter/leave, send/receive, collective communication, hardware performance counters, etc.) with specific properties distributed over multiple files.

OTF2 is a major part of run-time system in Score-P but can be also used as a separate software package because it is fast and platform independent. This tool also supports read/write APIs for C/C++/Python.

At the moment of writing this deliverable I/O support is planned but not currently supported. I/O tracing requires OTF 2.1 and a future version of Score-P (current version is 3.1). With I/O tracing support, using Score-P will be beneficial because it will allow high-level I/O to be captured and thus, will satisfy more requirements from deliverable D1.1 (for example CLM4 and CLM6). OTF 2.1 has support for the POSIX, ISO, MPI-IO, (P)NetCDF, HDF5 and ADIOS interfaces (see https://silc.zih.tu-dresden.de/otf2-2.1/group__io.html).

How to execute it

An example session of using Score-P in user script is like:

```
$ spack load -r mpi scorep
$ scorep mpicc -g mpi-speedup.c -o mpi-speedup mpif90
# Score-P environment variables
$ export SCOREP_ENABLE_TRACING=TRUE # enable tracing
$ export SCOREP_TOTAL_MEMORY=10000000 # trace buffer size
# export SCOREP_METRIC_PAPI=PAPI_FP_OPS
$ mpiexec -np 2 ./mpi-speedup
```

This will trace MPI calls and allow to distinguish application from communication calls. To understand internal behavior one has to use:

```
$ scorep --pdt mpicc -g mpi-speedup.c -o mpi-speedup
```

ScoreP comes with a few command line tools to explore the performance, for example, *scorep-score*:

```
$ scorep-score scorep-20170509_1259_6935968279002828/profile.cubex
Estimated aggregate size of event trace:                   2277 bytes
Estimated requirements for largest trace buffer (max_buf): 1139 bytes
Estimated memory requirements (SCOREP_TOTAL_MEMORY):       4097kB
(hint: When tracing set SCOREP_TOTAL_MEMORY=4097kB to avoid intermediate flushes or reduce
requirements using USR regions filters.)
flt    type max_buf[B] visits time[s] time[%] time/visit[us]  region
     ALL      1,138     52   10.24   100.0      196829.81  ALL
     MPI        852     30    0.03     0.3        1046.47  MPI
     USR        260     20    0.00     0.0           1.32  USR
     COM         26      2   10.20    99.7     5101864.89  COM
```

Additionally, the tool Cube can be used to explore profiles and Vampir [vamp] can be used to investigate traces. Both are graphical tools and require X11 forwarding. Examples of I/O analysis visualization on VampirTrace can be found on screenshots at page http://event.vampir.eu/sc12.

Smart Cities

Description

The proliferation of small sensors and devices around us capable of generating valuable information has created a new paradigm of computing known as Internet of Things. One of the most important concepts of this paradigm is the possibility of integrating and analysing all of this data input to make smart decisions in fields like healthcare, traffic management, water quality, air pollution and many more [Gubbi 2013].

Smart Cities are able to take advantage of these IoT networks to improve citizen's life. It can bring benefits to public transportation, garbage management, parking or education to name some examples [Zanella 2014]. The objective is to be able to gather data from different sensors located around the Smart City infrastructure and analyse it either at real time or offline. The information gathered can be used to improve services, make decisions or be published as open data for the citizens.

However, building this kind of infrastructure poses new challenges in many different levels. First the data has important contextual information like spatial or temporal that has to be considered at the moment of storing it. There are also privacy and security concerns that can be raised to protect the privacy of citizens. Also, the speed and volume of the data generated has to be considered. But most important is the heterogeneity of the different sensors and systems involved in this kind of infrastructure. For example, the communication protocol or the architecture of the sensors/devices can be different depending on its type creating the necessity of integrating all of the different sources and storing the data in our frontend.

Measurements and data coming from IoT devices are not only processed in the cloud, since the infrastructure and processing capabilities can be insufficient. The needs of, e.g., geographical distribution of resources, real-time communication, incorporation with large networks are handled by fog computing. Through this paradigm, part of processing is done by edge devices or clouds closer to data sources, resulting in less latency and bandwidth usage [Vaquero 2014].

Lastly the huge number of sensors and devices involved makes it difficult to detect failures and problems in the network, like disconnected wires, high and abnormal energy consumption or wrongly configured devices. The detection and root cause analysis of these events have to be considered to operate a Smart City effectively.

Smart City applications have been gaining more and more interest since 2015. However, there is a lot of research challenges that must be tackled before deploying effective solutions.

In this section, we introduce a set of benchmarks that are relevant for the smart cities topic. The selected benchmarks have been classified into several categories: stream processing benchmarks, batch processing benchmarks and others (combining both or focusing on other processing models). For each benchmark, we give a short description of the objectives, how we should execute it, describe the evaluated metrics and finally indicate which requirements it addresses with respect to deliverable D1.1. When relevant, we present some preliminary results we obtained through BigStorage past activities.

Linear Road (stream processing)

The Linear Road Benchmark [LinearRoad] simulates a toll system for the motor vehicle expressways of a large metropolitan area: "Every vehicle in Linear City is equipped with a sensor that emits a position report, which identifies the vehicle's exact coordinates every 30 seconds". Position reports are processed to generate statistics about traffic conditions on every segment of every expressway for every minute, including average vehicle speed, number of vehicles and existence of accidents. These statistics are used to determine toll charges for variable tolling. In addition, vehicles can issue queries to find out their current account balance with the expressway system, total tolls assessed on a given day and expressway, and travel time estimates. [LinearRoad]

"The Linear Road Benchmark makes it possible to compare performance characteristics of Stream Data Management Systems relative to each other and relative to alternative systems like Relational Databases. Stream Data Management Systems process streaming data by executing continuous historical queries while producing query results in real time."

Rationale

Linear Road is designed to measure how well a system can meet real-time query response requirements in processing high volume streaming and historical data. It has been endorsed by the developers of Aurora (out of Brandeis University, Brown University and MIT) and STREAM (out of Stanford University) as a basis for performance comparisons of stream processing approaches. It evaluates the requirements SCT1, SCT7 and SCT9, which have been defined in deliverable D1.1.

How to execute it

Details for running the Linear Road benchmark are described in [LinearRoad]. A set of resources for the historical data generator, the traffic simulator, configuration of the system running the benchmark including queries and the validator tools are available online at http://www.cs.brandeis.edu/~linearroad/. An implementation of the benchmark for [CQL] is also available online at http://infolab.stanford.edu/stream/cql-benchmark.html.

Section 3.3 Running the Benchmark in http://www.cs.brandeis.edu/~linearroad/linear-road.pdf explains the steps necessary to run the benchmark. Because "the Linear Road Benchmark makes it possible to compare performance characteristics of Stream Data Management Systems relative to each other and relative to alternative systems like Relational Databases (Stream Data Management Systems process streaming data by executing continuous historical queries while producing query results in real time)", all the steps and configuration aspects are important also in the Smart Cities context with respect to evaluating streaming technologies.

Metrics

The Linear Road benchmark is focused on the following metrics:

- Maximum and worst-case **response times** for toll notifications (e.g., queries response time)
- **Throughput** (processed tuples per second)
- **Scale factor:** "*the maximum scale factor at which the system can respond to the specified set of continuous and historical queries while meeting their response time and accuracy requirements. Each query answer must satisfy the response time and correctness requirements specified, and the throughput that a system can sustain in meeting these requirements constitutes the benchmark score (its L-Rating).*" [LinearRoad]

Preliminary results

In order to better understand the Linear Road benchmark the reader can explore [LRB2006] which focuses on latency-load measurements and effects of the network buffer size and data format on query response time.

Neptune (stream processing)

Neptune is a real-time stream processing engine [Neptune] designed to efficiently leverage the network I/O, CPU and memory resources in order to optimize throughput and scalability for stream handling. Neptune is a good fit for IoT data sizes (e.g., hundreds of bytes) and provides a benchmark with a set of stream jobs: we retain its generality for other IoT stream-based workloads and keep the proposed benchmark characteristics in [Neptune] under the same name in this document.

Rationale

"IoT encompasses settings where large numbers of ubiquitous sensors, actuators and embedded communication hardware are seamlessly integrated in the environment alongside information systems to store and process voluminous data produced by these monitored environments to form a communicating-actuating network. Middleware for on-demand storage and data analytics is considered a key element in an IoT reference architecture. Achieving high throughput stream processing in IoT and sensing settings involves challenges that impact the efficiency of network, CPU, and memory utilization" [Neptune].

Neptune provides a benchmark that considers the (network) buffer size and bandwidth usage with respect to throughput variations and end-to-end latency, making clear the importance of these aspects in a streaming benchmark. It evaluates requirements SCT1 and SCT5.

How to execute it

A diverse set of scenarios is described in [Neptune]. NEPTUNE was evaluated with a real-world stream processing application involving the monitoring of manufacturing equipment by processing data streams generated by sensors attached to the equipment in real time [DEBS]. The system ingests a continuous stream of readings captured by sensors.

Basically, one should configure the stream processing engine to be able to execute a number of stream jobs. E.g., if we evaluate the Neptune's metrics on Apache Flink, we follow the best of practice configuration settings and we run a cluster with a number of Flink nodes for up to a

maximum parallelism (https://ci.apache.org/projects/flink/flink-docs-master/). Then, we measure the cumulative throughput/cumulative bandwidth usage versus number of nodes (parallelism)/number of jobs. It is important to measure not only single node performance metrics (throughput versus message size/buffer size) but also cumulative metrics for the overall performance evaluation of a streaming engine dedicated to Smart Cities workloads.

Metrics

Neptune develops a benchmark to evaluate its framework and that is focused on the following metrics:

- **Throughput** versus Buffer Size
- **Latency** versus Buffer Size
- **Bandwidth** Usage versus Buffer Size
- **Cumulative throughput** and **cumulative bandwidth** usage with the number of concurrent jobs
- Throughput, end-to-end latency and bandwidth usage versus message size

Preliminary results

Neptune benchmark results are clearly presented in [Neptune]. We aim to reproduce these benchmarks results and use them to further evaluate our prototype Kera [KERA].

RIoTBench (stream processing)

The Real-time IoT Benchmark suite has been developed to evaluate Data Streaming Processing Systems (DSPS) for streaming IoT applications. The benchmark includes 27 common IoT tasks classified across various functional categories and implemented as reusable micro-benchmarks. It also includes four IoT application benchmarks composed from these tasks, and that leverage various dataflow semantics of DSPS. The applications are based on common IoT patterns for data pre-processing, statistical summarization and predictive analytics. These are coupled with four stream workloads sourced from real IoT observations on smart cities and fitness, with peak streams rates that range from 500 − 10, 000 messages/sec and diverse frequency distributions.

Rationale

Urban IoTs architectures provide the foundations for Smart City applications. Within this stack, Distributed Stream Processing Systems (DSPS) are a key element to analyse and make decisions in real time based on the data coming from multiple sources. IoT applications, and consequently many Smart City applications, are composed of a set of representative tasks for data pre-processing, summarisation/aggregation and predictive analytics among others. RIoTBench is a benchmarking tool to execute many of these representative tasks, either as a user built workflow or a full application, to evaluate the performance of DSPS. It relates to requirements SCT1, SCT2 and SCT9.

How to execute it

RIoTBench only works with Storm [Storm]. Once you compile with maven the package you can submit a task to storm with the following command:

```
storm jar
<stormJarPath>   in.dream_lab.bm.stream_iot.storm.topo.micro.Micr
oTopologyDriver  C  <microTaskName>  <inputDataFilePath used by
CustomEventGen and spout>   PLUG-<expNum>  <rate as
1x,2x>  <outputLogPath>   <tasks.properties File
Path>   <TopoName>
```

Where:

- microTaskName is one of the micro tasks available
- inputDataFilePath: the datafile that will be streamed
- expNum: *An experiment number*
- rate: the rate in which data should be generated
- outputLogPath: The log where all the metrics will be dumped
- task.properties File Path: A file with the different properties of that particular tasks
- TopoName: It defines the connections between the different bolts.

There are also full applications ready to be executed. An example of a smart cities use case would be a taxi GPS processing worklow such as the IoTPredictionTopologyTAXI.java:

```
storm jar
<stormJarPath>   in.dream_lab.bm.stream_iot.storm.topo.apps.IoTPredicti
onTopologyTAXI
```

More information is provided in the paper [shukla2017riotbench]

Metrics

RIoTBench uses the following metrics:

- **Latency** - The latency of that task to process a message coming from a sensor. The latency per message may vary depending on the input rate, resources allocated to the task, and the type of message being processed
- **Throughput** - The output throughput is the aggregated rate of output messages emitted out of the tasks, measured in messages per second. The throughput of a dataflow depends on the input throughput. It is also useful to measure the peak throughput that can be supported by a given application, which is the maximum stable rate that can be processed using a fixed quanta of resources.
- **Jitter**: The ideal output throughput may deviate due to variable rate of the input streams, change in the paths taken by the input stream through the dataflow (e.g., at a Hash pattern), or performance variability of the DSPS. We use jitter to track the variation in the output throughput from the expected output throughput.
- **CPU and Memory Utilization:** Streaming IoT dataflows are expected to be resource intensive, and the ability of the DSPS to use the distributed resources efficiently with minimal overhead is important. This also affects the VM resources and consequent price to be paid to run the application using the given stream processing platform.

YCSB (stream processing)

Yahoo Cloud Service Benchmark (YCSB) is a standard framework to assist in the evaluation of different cloud systems. Its main objective is to model systems based on online read and write access to data, such as web services: photo tagging, online stores, post threads, etc. The benchmarking framework provides a set of workloads that generates client requests to different back ends like Cassandra, RAMCloud, HBase or MongoDB. These workloads have different read and write ratios and different ways of accessing the data (e.g. scan records, one record at a time...). YCSB is easily customisable and allows end-users to define new types of workloads and to configure the clients with different parameters. Finally, it offers the possibility to add new back ends by simply implementing the requested interface.

Rationale

"The goal of the YCSB project is to develop a framework and common set of workloads for evaluating the performance of different "key-value" and "cloud" serving stores." [YCSBWiki]. Cloud services will be an essential part of Smart citizen's life ranging from open data access API`s to different web services related to health, social networking or public services. YCSB allows us to evaluate this aspect of Smart Cities applications, in particular requirements SCT1, SCT2, SCT5 and SCT10.

How to execute it

Step 1. Set up the database (e.g., Cassandra, RAMCloud)

The database needs some preconfiguration in order for the clients to insert the records. Refer to the specific preparations depending on the DB in https://github.com/brianfrankcooper/YCSB/wiki/Using-the-Database-Libraries

Step 2. Choose the appropriate workload

in https://github.com/brianfrankcooper/YCSB/tree/master/workloads there are a set of workload templates that can be used. This will be specified later on in the ycsb command

Step 3. Choose the runtime parameters (the number of threads, the target number of operations per second, enabling status reports etc.).

Step 4. Load the data.

There are two phases in YCSB. Loading data phase and the transactions phase. To load it you use the load YCSB command with a parameter that depends on the database that we want to use e.g. Cassandra, Hbase.

Some specific databases require specific parameters that are also included in https://github.com/brianfrankcooper/YCSB/wiki/Using-the-Database-Libraries

Step 5. Execute the workload.

Same as the previous step but executing the run command. More details about running the benchmark are available in https://github.com/brianfrankcooper/YCSB/wiki/Running-a-Workload#step-6-execute-the-workload. At the end of the execution, the user gets a report of the execution in terms of the metrics that we present in the following section

An example of a use case would be to simulate several clients that try to access a CassandraDB with information about bus itineraries.

Load data from workload C which is read only

```
ycsb-0.12.0/bin/ycsb.sh load cassandra-cql -P ycsb-
0.12.0/workloads/workloadc -p hosts=<cassandra_hosts>

ycsb-0.12.0/bin/ycsb.sh run cassandra-cql -P ycsb-
0.12.0/workloads/workloadc -p hosts=<cassandra_hosts>
```

Metrics

YCSB is focusing on the following metrics:

- **Throughput(ops/sec):** The number of operations per second that the client has managed to perform
- **RunTime(ms):** The time it took to the client to complete the workload and insert a given number of records
- **Latency Statistics (min,Max,Average):** Descriptive Statistics about the latency of each operation

Preliminary results

YCSB was used in [ICDCS-RC] to reveal shortcomings in the RAMCloud storage systems, more specifically in the replication. By combining simple (read-only) and mixed (read-write) workloads, the authors the authors were able to identify a baseline of RAMCloud's performance then discover performance bottlenecks when replication was enabled in the system. Additionally, it was used to evaluate the overhead and non-intrusiveness of a monitoring tool designed specifically for fog systems, which are an essential part of smart citiy infrastructures. By executing the different YCSB workloads we could evaluate how a flexible and decentralized monitoring system impacted positively in the number of operations per second compared to a typical centralised monitoring system.

BigBench (Batch Processing)

BigBench is an end-to-end Big Data benchmark proposal based on a fictitious retailer who sells products to customers via physical and online stores [BigBench]. The proposal covers a data model, synthetic data generator and workload descriptions. It also includes directions for big data metrics specific to data loading and workload execution.

Rationale

The benchmark proposal covers a data model and synthetic data generator that addresses the variety, velocity and volume aspects of big data systems containing structured, semi-structured and unstructured data [BigBench]. Additionally, the data generator provides scalable volumes of raw data based on a scale factor. This allows BigBench to be used to test the batch processing aspects of Smart City applications, and more specifically requirements SCT4 and SCT10.

How to execute it

The original proposal includes workloads in the form of 30 queries expressed in plain English. The queries are divided into four business categories (Marketing, Merchandising, Operations and Supply chain) and organized along three technical dimensions:

1. Query processing type: declarative, procedural and hybrid.
2. Data sources: structured, semi-structured and unstructured.
3. Analytic techniques: statistics analysis, reporting and data mining.

The benchmark has been initially tested on the Teradata Aster Database (TAD) by generating and loading a 200 Gigabyte data set and executing the queries mentioned above, implemented using TAD's SQL-MR syntax [BigBench]. The benchmark specification can also be implemented on any database or storage system relevant to Smart City scenarios. In particular, we plan to implement it in order to evaluate the storage solutions developed within BigStorage.

Metrics

The final design for the benchmark metrics is deferred to future work. However, an initial set includes the following metrics:

- Initial load time
- Data refresh time
- Query execution time

COSBench (Batch Processing)

COSBench is an open-source benchmarking tool used to measure Cloud Object Storage Service (COSS) performance. It is using the S3 or Swift API. The benchmark is meant to be used by cloud solution providers to compare different hardware/software stacks as well as to identify bottlenecks and make performance optimization. It also offers cross-platform deployment and pluggable adapters for different storage systems.

Rationale

In the context of Big Storage, using COSBench to benchmark a data storage system like the Fujitsu CD10000, on how it is handling a huge set of very small files (like the IOPS used in Smart Cities applications) reveals a set of insights on the specific issues of dealing with stream data at scale.

As there are storage requirements for Smart Cities data, CosBench can benchmark the system for requirement SCT5. The scalability of the system is also important, as data generated will grow

more, making requirement SCT10 (Scalable System) very important. Data Locality (SCT8) will also play an important role for Smart Cities, where CosBench can simulate a local system that stores all the data from a Smart City application.

How to execute it

The benchmark creates different workloads from a workload model that can be configured in terms of the following attributes:

- Concurrency pattern: worker number and container range.
- Access pattern: object size and read/write ratio.
- Usage limitation: operation count and running time.

Metrics

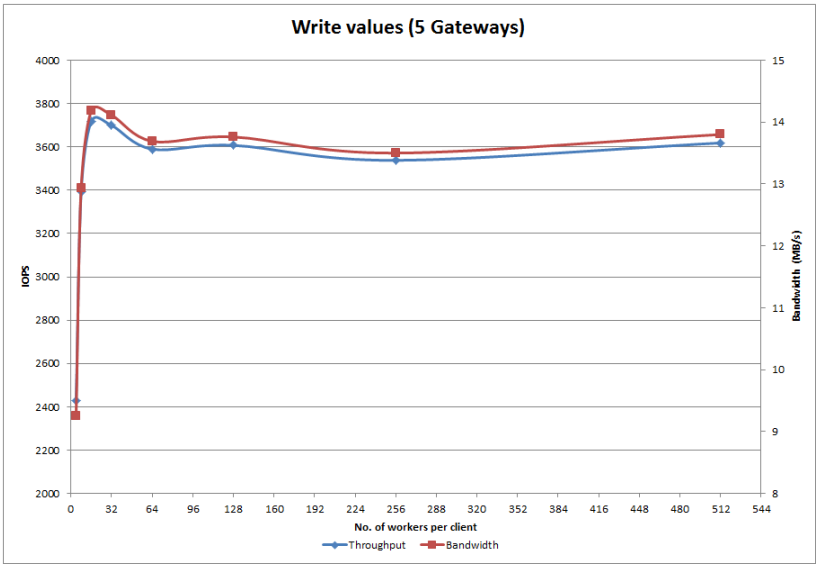
- The benchmark is currently capable of measuring mainly 3 performance metrics:
- **Response time:** duration between operation initiation and completion.
- **Throughput:** total number of operations performed per second.
- **Bandwidth:** total amount of data transferred per second.

Preliminary Results

A benchmark was performed in Fujitsu Labs. The setup was:

- 4 and 5 Ceph Object Gateways
- An Eternus CD10000 system with 4 storage nodes
- Ceph Storage
    - Replicated Pools
    - Erasure Coded Pools

The following results (Figure 10) are from testing object sizes of 4K.



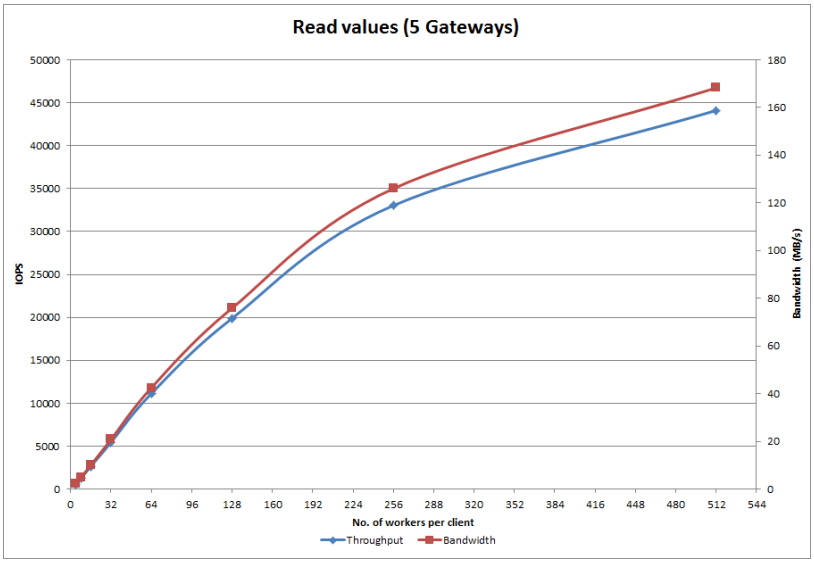


Figure 10 Test results from Cosbench

The results showed possible bottlenecks on performance, offering research possibilities on performance improvement.

IoTABench

The IoT Analytics Benchmark (IoTABench) is a benchmark toolkit that facilitates repeatable testing and can be easily extended to multiple IoT use cases [IoTABench]. It consists of three components: a scalable synthetic data generator, a set of SQL queries and a test harness. The initial implementation of IoTAbench focuses on a single use case: smart metering of electrical meters and the development of additional use cases is left for future work.

Rationale

The aim of IoTABench is to bridge the divide between Big Data research and practice by providing practitioners with insights into a platform managing sensor data at production scale [IoTABench]. To that end, the benchmark generates a large data set (22.8 trillion distinct readings, or 727 TB of raw data) with realistic properties using a Markov chain-based synthetic data generator. Hence, this benchmark can be used to evaluate real-life Smart Cities scenarios and helps in testing requirements SCT4, SCT5 and SCT10.

How to execute it

The benchmark was run by its authors on a cluster of 8 HP Proliant DL380p servers with HP Vertica Analytics platform 7.0 [HPVertica]. The smart metering use case was used in the experiments and the following steps were followed:

- Generate the synthetic dataset (40 million sensors with a 10 minutes reading interval)
- Load the raw data into a staging area
- Repair the raw data and store the repaired data (1% “missing” readings)

- Delete the raw data and empty the staging area
- Analyse the repaired data: the workload includes the following queries written in SQL:
    - Total readings
    - Total consumption
    - Peak consumption
    - Consumption time-series
    - Top consumers
    - Time of usage billing

While the generation of data (step 1) and the analysis queries (step 5) are portable across SQL-enabled Big Data platforms, the queries to load, repair and optimize the data (steps 2, 3 and 4) use special features of HP Vertica Analytics Platform [IoTABench] and hence should be rewritten when benchmarking other platforms (i.e. the ones designed in the context of BigStorage).

Metrics

- **Data loading and repairing:** measures the time needed to load and repair the data using two different strategies: scale-up and scale-out.
- **Analysis performance:** measures the time needed to execute each of the workload queries.

RadosBench (Batch Processing)

Ceph comes with an inbuilt benchmarking program, known as RADOS bench, which is used to measure the performance of a Ceph object store. We make use of RADOS bench to get the performance metrics of the CD10000 Ceph cluster. Good performance results can be achieved if someone is using recommended hardware with performance-tuned Ceph deployment, like the CD10000 system from Fujitsu.

Rationale

As there are storage requirements for Smart Cities data, RadosBench can benchmark the system for requirement SCT5. The scalability of the system is also important, as data generated will grow more, making requirement SCT10 (Scalable System) very important. Data Locality (SCT8) will also play an important role for Smart Cities, where RadosBench can simulate a local system that stores all the data from a Smart City application.

How to execute it

- rados bench -p my_pool 300 write
- rados bench -p my_pool 300 seq
- API used: Native Ceph Object without the without the S3/Swift API.

All measurements were performed with the CEPH inbuilt RadosBench benchmarking program.

- Software : Red Hat Ceph Storage 2.0 (Ceph Jewel release 10.2.2)
- Hardware : ETERNUS CD10000 v2.1, based on Red Hat Enterprise Linux 7.2
- Number of Gateway Clients : 4 or 5
- Ceph Cluster :

  - 66 OSDs (6 storage nodes)
  - 55 OSDs (5 storage nodes)
  - 44 OSDs (4 storage nodes)
- Replicated Pool Settings : 3 Replicas, Placement Groups = 2048
- EC Pool Settings :
  - k=3, m=2
  - k=4, m=2
- Network : 20Gbit/s, two stacked Brocade switches, MTU = 7500
- Ceph scrubbing and deep scrubbing disabled during the measurements
- Sets of workers per client tested: 8, 16, 32, 64, 128, 256

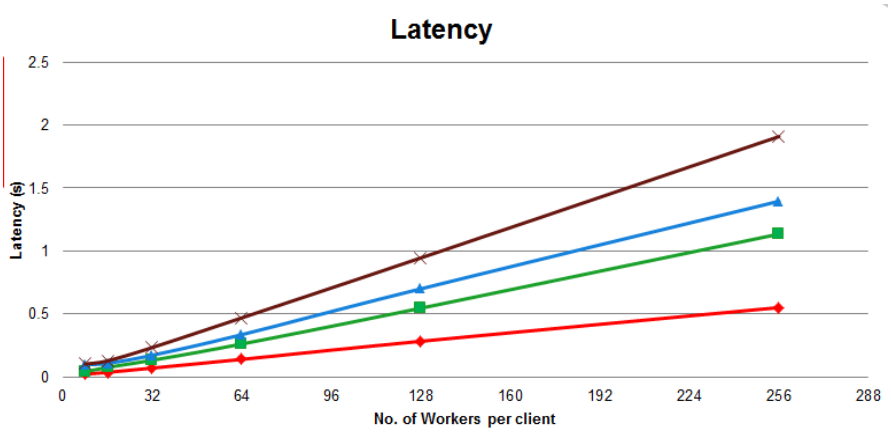
Metrics

- Latency (in seconds)
- Throughput (in IOPS)
- Bandwidth (in MB/s)

Preliminary results

The tests were performed in order to test the CD10000 with typical Smart Cities workloads.

The following results show a comparison between different number of gateways and different data storage technologies, using 4M files and sequential READ.



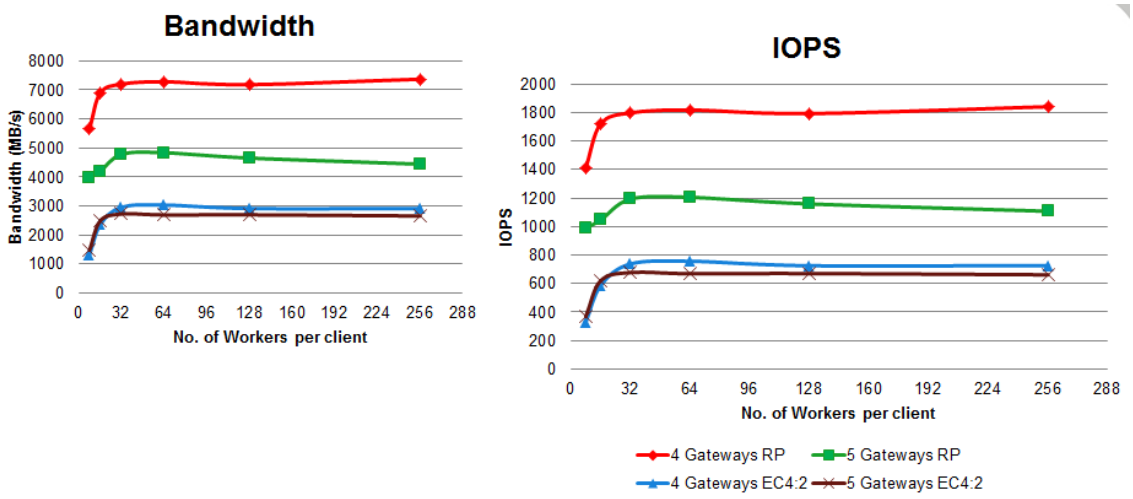


Figure 11 Performance results from Fujitsu Eternus CD10000 system, using RadosBench

Figure 11 show the comparison between different storage technologies used on the Fujitsu CD10000 Ceph system. The workload showed that, by increasing the workers above 64, there is no increase in the performance. The results show very good performance values, making CEPH and Fujitsu hardware a very relevant system for a typical Smart City workload.

BigDataBench (Other)

As a multi-discipline research and engineering effort, i.e., system, architecture, and data management, from both industry and academia, BigDataBench is an open-source big data benchmark suite. The current BigDataBench 3.2 version models five typical and important big data application domains: search engine, social networks, e-commerce, multimedia analytics, and bioinformatics. In total, it includes 14 real-world data sets, and 34 big data workloads. In general, it provides an interesting set of workloads in both streaming and batch implementations

Rationale

As a multi-discipline research and engineering effort, i.e., system, architecture, and data management, from both industry and academia, BigDataBench is an open-source big data benchmark suite. The current BigDataBench 3.2 version models five typical and important big data application domains: search engine, social networks, e-commerce, multimedia analytics, and bioinformatics. In total, it includes 14 real-world data sets, and 34 big data workloads. In general, it provides an interesting set of workloads in both streaming and batch implementations. Requirements tested: Batch processing (SCT4), Streaming (SCT1, SCT5), Interactive queries (SQL like queries) (SCT3).

How to execute it

Requirements packages to install:

- Java JDK: version 1.6 or later
- OpenSSH Hadoop (recommend version 1.2.1)

Install the BigData tools to be benchmarked: Technologies that it uses: Hadoop, Spark, Flink, MPI, Spark Streaming, Nutch, JStorm, HBase, MySQL, GraphX, Flink Gelly, GraphLab, Hive, Shark, Impala

Install the corresponding workload to run, follow the instruction from: http://prof.ict.ac.cn/wp-content/uploads/2014/12/BigDataBench-User-Manual.pdf

Metrics

- **Latency**: How much does it take to do batch processing of data to build machine learning models. This models can be used to predict city phenomena like traffic in a given area or energy consumption.
- **Data Throughput**: What's the data throughput of the stream analysis engine. It is important that the system can cope with the constant stream of information coming from the sensor network installed throughout the city. This can be tested in different scenarios:
    - Storing the stream of data in a NoSQL database like MongoDB or Cassandra
    - Applying a filter to the data, transforming it or discarding some values
    - Using the stream of data as an input to a machine learning model
    - A combination of the above

A possible drawback is, that the huge rate of data generated by these sensors cannot be simulated with this benchmark.

HiBench (Other)

HiBench is a big data benchmark suite that helps evaluate different big data frameworks in terms of speed, throughput and system resource utilizations. It contains a set of Hadoop, Spark and streaming workloads, including Sort, WordCount, TeraSort, Sleep, SQL, PageRank, Nutch indexing, Bayes, Kmeans, NWeight and enhanced DFSIO, etc. It also contains several streaming workloads for Spark Streaming, Flink, Storm and Gearpump.

Rationale

HiBench is a big data benchmark suite that helps evaluate different big data frameworks in terms of speed, throughput and system resource utilizations. It contains a set of Hadoop, Spark and streaming workloads, including Sort, WordCount, TeraSort, Sleep, SQL, PageRank, Nutch indexing, Bayes, Kmeans, NWeight and enhanced DFSIO, etc. It also contains several streaming workloads for Spark Streaming, Flink, Storm and Gearpump. It evaluates requirements SCT1, SCT2, SCT4, SCT5 and SCT10.

How to execute it

A. Building the benchmark

1. Build All
   `mvn -Dspark=2.1 -Dscala=2.11 clean package`
2. Build a specific framework benchmark
   HiBench 6.0 supports building only benchmarks for a specific framework. For example, to build the Hadoop benchmarks only, we can use the below

command:
mvn -Phadoopbench -Dspark=2.1 -Dscala=2.11 clean package

3. To build Hadoop and Spark benchmarks
   mvn -Phadoopbench -Psparkbench -Dspark=2.1 -Dscala=2.11 clean package
   Supported frameworks includs: hadoopbench, sparkbench, flinkbench, stormbench, gearpumpbench.
4. Specify Scala Version
   mvn -Dscala=2.10 clean package
5. Specify Spark Version
   mvn -Psparkbench -Dspark=1.6 -Dscala=2.11 clean package
6. Build a single module
   If you are only interested in a single workload in HiBench, you can build a single module. For example, the below command only builds the SQL workloads for Spark.
   mvn -Psparkbench -Dmodules -Psql -Dspark=2.1 -Dscala=2.11 clean package
   Supported modules includes: micro, ml(machine learning), sql, websearch, graph, streaming, structured Streaming (spark 2.0 or 2.1).
7. Build Structured Streaming
   mvn -Psparkbench -Dmodules -PstructuredStreaming clean package

B. Running the benchmark

1. HadoopBench
   Follow the instruction to run on: https://github.com/intel-hadoop/HiBench/blob/master/docs/run-hadoopbench.md
2. SparkBench
   Follow the instruction to run on: https://github.com/intel-hadoop/HiBench/blob/master/docs/run-sparkbench.md
3. StreamingBench
   Follow the instruction to run on: https://github.com/intel-hadoop/HiBench/blob/master/docs/run-streamingbench.md

Metrics

- **Latency**: How much does it take to do batch or stream processing of data to build machine learning models or extract meaningful patterns from the data.
- **Data Throughput**: What is the data throughput of the stream analysis engine. It is important that the system can cope with the constant stream of information coming from the sensor network installed throughout the city. Also, impact of particular parts of a system on data throughput can be measured. For instance, it enables to detect bottlenecks in network and processing units.

Preliminary results

In order to better understand the behaviour of two Big Data processing engines, Spark and Flink, and to quantify the differences in the design choices, we run a series of extensive experiments involving representative batch and iterative workloads described also in HiBench (e.g.,

WordCount, Grep TeraSort, KMeans, PageRank, Connected Components) [SparkVersusFlink]. These benchmarks focus on the end-to-end execution time metric correlated to different parameter settings (e.g., parallelism, memory management), the resource usage (CPU, memory, network, disk), and in the context of strong and weak scalability. The reader can further explore the experimental settings and benchmark results in [SparkVersusFlink] where a summary of insights is provided.

SparkBench (Other)

Spark-Bench is a benchmarking suite specific for Apache Spark. It comprises a representative and comprehensive set of workloads belonging to four different application types that currently supported by Apache Spark, including machine learning, graph processing, streaming and SQL queries. The chosen workloads exhibit different workload characteristics and exercise different system bottlenecks; currently we cover CPU, memory, and shuffle and IO intensive workloads.

Rationale

Sparkbench covers four categories of workloads that are present in any data processing use case, included Smart Cities:

- Machine Learning: Image recognition, self-driving transport
- Graph Computation: traffic control, social networking
- SQL query: Access to data by citizens, reporting
- Streaming applications: Power Grid, Smart Traffic Ligths

It evaluates requirements SCT1, SCT2, SCT4, SCT5 and SCT10.

How to execute it

The JAR needs to be built, in order to use the benchmark.

```
git clone https://github.com/synhershko/wikixmlj.git
cd wikixmlj
mvn package install
```

Then we have to configure several parameters needed to connect to Spark and HDFS to store the data. A template is available at `<SparkBench_Root>/conf/env.sh.template.`

The important variables that must be set are:

SPARK_HOME The Spark installation location

HADOOP_HOME The HADOOP installation location

SPARK_MASTER Spark master

HDFS_MASTER HDFS master

Once the configuration has been done we can start executing the workloads. First, we need to generate the data and then we can run the workload:

<SPARK_BENCH_HOME>/<Workload>/bin/gen_data.sh

<SPARK_BENCH_HOME>/<Workload>/bin/run.sh

For streaming applications, we need the data producer and the consumer. This need to be executed in parallel in two different terminals or processes.

<SPARK_BENCH_HOME>/Streaming/bin/gen_data.sh <Application>

<SPARK_BENCH_HOME>/Streaming/bin/run.sh <Application>

Where application can be one of the available apps like TwitterPopularTags ir PageViewStream

The results will be available in <SPARK_BENCH_HOME>/report

In general, there are 3 different files under one workload folder that can be customised by the user

<Workload>/bin/config.sh change the workload specific configurations

<Workload>/bin/gen_data.sh

<Workload>/bin/run.sh

Metrics

The metrics of SparkBench reflect performance:

- **Job execution time (seconds)** it measures the job execution time of each workload. This reflects performance
- **Data process rate (MB/second)** defined as input data size divided by job execution time. This reflects the data processing capability.

Preliminary results

SparkBench was used to evaluate the gains on performance of a model that decided an optimal task parallelism level in an Apache Spark deployment. By measuring the time, it took to complete the different workloads of the benchmark, we could make a comparison between our model and a default parallelism level [hernandez2017using].

Link to ESRs

In this section, we present the link between the ESRs as the benchmarks defined by each use case.

ESR	HBP	SKA	Climate	Smart cities
ESR1			x	x
ESR2				x
ESR3			x	x
ESR4	x		x	
ESR5				x
ESR6				
ESR7			x	
ESR8	x		x	x
ESR9			x	
ESR10	x			
ESR11	x			
ESR12			x	
ESR13				
ESR14		x	x	x
ESR15				x

Conclusion

This deliverable presents a set of benchmarks that represent the four use cases: The Human Brain project, the Climate modelling, the Square Kilometre Array, and smart cities. These benchmarks have been defined in enough detail for ESRs to use them to test their progress.

For each initiative and benchmark, we have detailed why it is important (including the requirements they fulfil), how to use and parametrize the benchmark, as well as some initial results that can help to understand progress of ESRs in the near future.

In order to model the different use cases, we have defined (or adopted) 1 benchmark for the human brain project, 8 benchmarks for square Kilometre array, 5 benchmarks for climate, and 9 benchmarks for smart cities. With these 23 benchmarks, we cover most for the relevant requirements defined in D1.1.

Finally, we have specified what ESRs may benefit from what use cases (and set of benchmarks) to show their progress.

References

[bbfs_git] https://github.com/pierreis/bbfs

[bbfs_paper] Could Blobs Fuel Storage-Based Convergence Between HPC and Big Data? (Pierre Matri, Yevhen Alforov, Alvaro Brandon, Michael Kuhn, Philip Carns, Thomas Ludwig), In 2017 IEEE International Conference on Cluster Computing, pp. 81–86, IEEE Computer Society, CLUSTER 2017, Honolulu, USA, 2017-09

[benchio] https://github.com/EPCCed/benchio

[BigBench] Ghazal, T. Rabl, M. Hu, F. Raab, M. Poess, A. Crolotte and H.-A. Jacobsen. *Bigbench: Towards an industry standard benchmark for big data analytics*. In Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. ACM (2013). pp. 1197–1208.

[blast] "mpiBLAST: Open-Source Parallel BLAST," 2017, http://www.mpiblast.org

[CQL] Arvind Arasu, Shivnath Babu, and Jennifer Widom. 2006. The CQL continuous query language: semantic foundations and query execution. *The VLDB Journal* 15, 2 (June 2006), 121-142. DOI=http://dx.doi.org/10.1007/s00778-004-0147-z

[DEBS] http://debs.org/debs-2012-grand-challenge-manufacturing-equipment/

[ecoham] U. H. Institute of Oceanography, "ECOHAM," https://wiki.zmaw.de/ifm/ECOHAM

[fuse] https://github.com/libfuse/libfuse

[g5k] https://www.grid5000.fr/mediawiki/index.php/Grid5000:Home

[hdf5] HDF5, https://support.hdfgroup.org/HDF5

[hernandez2017using] Hernández, Álvaro Brandón, et al. "Using machine learning to optimize parallelism in big data applications." Future Generation Computer Systems (2017).

[HPVertica] HPE Vertica Analytics Platform Overview

[ICDCS-RC] Y. Taleb, S. Ibrahim, G. Antoniu and T. Cortes, "Characterizing Performance and Energy-Efficiency of the RAMCloud Storage System," *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, Atlanta, GA, 2017, pp. 1488-1498.

[IOR-FLAGS] https://github.com/LLNL/ior/blob/master/doc/USER_GUIDE

[IOR-LANL] https://github.com/IOR-LANL/ior

[IOR-LLNL] https://github.com/LLNL/ior

[IoTABench] M. Arlitt, M. Marwah, G. Bellala, A. Shah, J. Healey and B. Vandiver. *IoTABench: An internet of things analytics benchmark*. In Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering, ICPE '15. ACM (2015). pp. 133–144.

[Ippen2017] Ippen, T., Eppler, J.M., Plesser, H.E. and Diesmann, M., 2017. Constructing neuronal network models in massively parallel environments. Frontiers in neuroinformatics, 11.

[KERA] Ovidiu-Cristian Marcu, Alexandru Costan, Gabriel Antoniu, María S. Pérez-Hernández. Kera: A Unified Storage and Ingestion Architecture for Efficient Stream Processing, 2017, https://hal.inria.fr/hal-01532070/

[Kunkel2014] Kunkel, S., Schmidt, M., Eppler, J.M., Plesser, H.E., Masumoto, G., Igarashi, J., Ishii, S. Fukai, T., Morrison, A., Diesmann, M. and Helias, M., 2014. Spiking network simulation code for petascale computers. Frontiers in neuroinformatics, 8

[Kunkel2017] Kunkel, S. and Schenck, W., 2017. The NEST Dry-Run Mode: Efficient Dynamic Analysis of Neuronal Network Simulation Code. Frontiers in neuroinformatics, 11, p.40.

[LinearRoad] Arvind Arasu, Mitch Cherniack, Eduardo Galvez, David Maier, Anurag S. Maskey, Esther Ryvkina, Michael Stonebraker, and Richard Tibbetts. 2004. Linear road: a stream data management benchmark. In *Proceedings of the Thirtieth international conference on Very large data bases - Volume 30* (VLDB '04), Mario A. Nascimento, M. Tamer Özsu, Donald Kossmann, Renée J. Miller, José A. Blakeley, and K. Bernhard Schiefer (Eds.), Vol. 30. VLDB Endowment 480-491.

[LRB2006] Navendu Jain, Lisa Amini, Henrique Andrade, Richard King, Yoonho Park, Philippe Selo, and Chitra Venkatramani. 2006. Design, implementation, and evaluation of the linear road benchmark on the stream processing core. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data* (SIGMOD '06). ACM, New York, NY, USA, 431-442. DOI=http://dx.doi.org/10.1145/1142473.1142522

[macsio_design] https://codesign.llnl.gov/macsio_doc/macsio_design_intro_final_html/macsio_design_intro_final.html

[macsio_doc] https://codesign.llnl.gov/macsio_doc/doxyout/html/index.html

[macsio_git] https://github.com/LLNL/MACSio

[macsio_llnl] https://codesign.llnl.gov/macsio.php

[macsio_paper1] Dickson, James, Wright, Steven A., Maheswaran, Satheesh, Herdman, J. A., Harris, Duncan, Miller, Mark C. and Jarvis, Stephen A. (2017) Enabling portable I/O analysis of commercially sensitive HPC applications through workload replication. In: Cray User Group 2017, Redmond, California, USA, 7-12 May 2017. Published in: Cray User Group 2017 Proceedings (CUG2017 Proceedings) pp. 1-14.

[macsio_vi4io] https://www.vi4io.org/tools/benchmarks/macsio

[MOGON] High Performance Computing, MOGON, https://hpc.uni-mainz.de/

[mom] "Ocean circulation models," 2017, https://www.gfdl.noaa.gov/ocean-model

[mpi] MPI Forum, http://mpi-forum.org

[nbench_git] NetCDF-Bench source, https://github.com/joobog/netcdf-bench

[nbench_vi4io] NetCDF-Bench, https://www.vi4io.org/tools/benchmarks/netcdf-bench

[Neptune] T. Buddhika S. Pallickara "NEPTUNE: Real time stream processing for internet of things and sensing environments" Proc. IEEE Int. Parallel Distrib. Process. Symp. pp. 1143-1152 2016.

[NEST] “The NEST Simulator”, http://www.nest-simulator.org/

[netcdf] NetCDF, https://www.unidata.ucar.edu/software/netcdf/

[otf2] Open Trace Format http://www.paratools.com/otf/

[piohpc] http://www.prace-ri.eu/IMG/pdf/WP236.pdf

[raytracing] L. Wang, J. Zhan, C. Luo, Y. Zhu, Q. Yang, Y. He, W. Gao, Z. Jia, Y. Shi, S. Zhang, C. Zheng, G. Lu, K. Zhan, X. Li, and B. Qiu, “BigDataBench: A big data benchmark suite from Internet services,” in 2014 IEEE 20th International Symposium on High Performance

[scorep_chapt] Andreas Knüpfer, Christian Rössel, Dieter an Mey, Scott Biersdorff, Kai Diethelm, Dominic Eschweiler, Markus Geimer, Michael Gerndt, Daniel Lorenz, Allen D. Malony, Wolfgang E. Nagel, Yury Oleynik, Peter Philippen, Pavel Saviankou, Dirk Schmidl, Sameer Shende, Ronny Tschüter, Michael Wagner, Bert Wesarg, Felix Wolf: Score-P: A Joint Performance Measurement Run-Time Infrastructure for Periscope, Scalasca, TAU, and Vampir. Parallel Tools Workshop 2011: 79-91 https://link.springer.com/content/pdf/10.1007%2F978-3-642-31476-6_7.pdf
[cube] https://apps.fz-juelich.de/scalasca/releases/cube/4.3/docs/manual/cube4api.html

[scorep_paper] A. Knupfer, C. Rossel, D. Mey, S. Biersdorff, K. Diethelm, ¨ D. Eschweiler, M. Geimer, M. Gerndt, D. Lorenz, A. Malony, W. E. Nagel, Y. Oleynik, P. Philippen, P. Saviankou, D. Schmidl, S. Shende, R. Tschuter, M. Wagner, B. Wesarg, and F. Wolf, “Score-p: A ¨ joint performance measurement run-time infrastructure for periscope, scalasca, tau, and vampir,” in Tools for High Performance Computing 2011, 2012. [Online]: http://dx.doi.org/10.1007/978-3-642-31476-6 7

[scorep_prs] https://www.olcf.ornl.gov/wp-content/uploads/2015/02/Score-P-OLCF-Tutorial.pdf

[scorep] http://www.vi-hps.org/projects/score-p/

[shukla2017riotbench] Shukla, Anshu, Shilpa Chaturvedi, and Yogesh Simmhan. "RIoTBench: A Real-time IoT Benchmark for Distributed Stream Processing Platforms." arXiv preprint arXiv:1701.08530 (2017).

[spack] Spack, https://spack.io/

[SparkVersusFlink] Ovidiu-Cristian Marcu, Alexandru Costan, Gabriel Antoniu, María S. Pérez-Hernández. Spark versus Flink: Understanding Performance in Big Data Analytics Frameworks. In: Cluster 2016—The IEEE 2016 International Conference on Cluster Computing, Taipei, Taiwan. https://hal.inria.fr/hal-01347638/document

[Storm] http://storm.apache.org/

[toolsguide] http://www.vi-hps.org/upload/material/general/ToolsGuide.pdf

[vamp] https://www.vampir.eu/tutorial/manual/getting_started

[YCSBWiki] https://github.com/brianfrankcooper/YCSB/wiki

[zlib] ZLIB, https://zlib.net