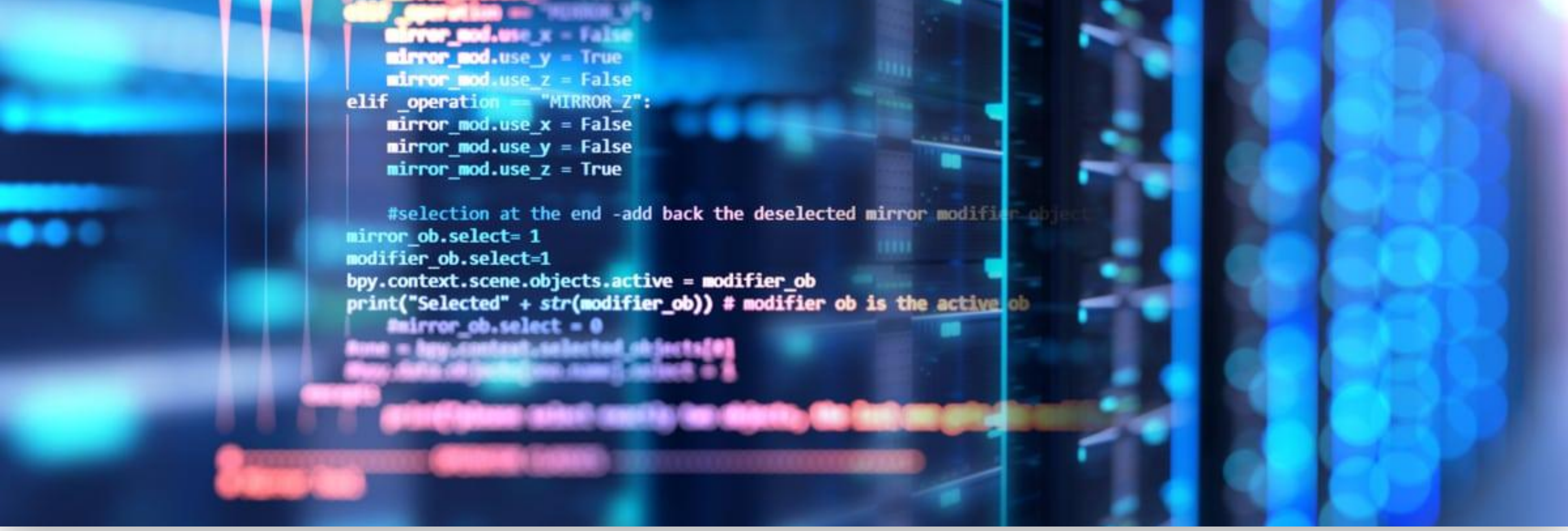


Apache Spark over Frontera cluster

Earlier Experiences

BIG HPC HIGH PERFORMANCE COMPUTING

TACC TEXAS ADVANCED COMPUTING CENTER

Cofinanciado por: COMPETE 2020 Lisb@20[20] PORTUGAL 2020 UNIÃO EUROPEIA Fundo Europeu de Desenvolvimento Regional

FCT Fundação para a Ciência e a Tecnologia

UTAustin Portugal

Outline

- Motivation
- Overview
- Frontera
- Apache Spark
- Benchmark
- Performance optimization
- Thanks

Motivation

- Bring the popular Apache Spark processing engine to HPC
- Run the software without any cluster side setup requirements
- Bypass rootless execution environments limitations
- Provide an optimized Apache Spark configuration for Frontera cluster

Overview

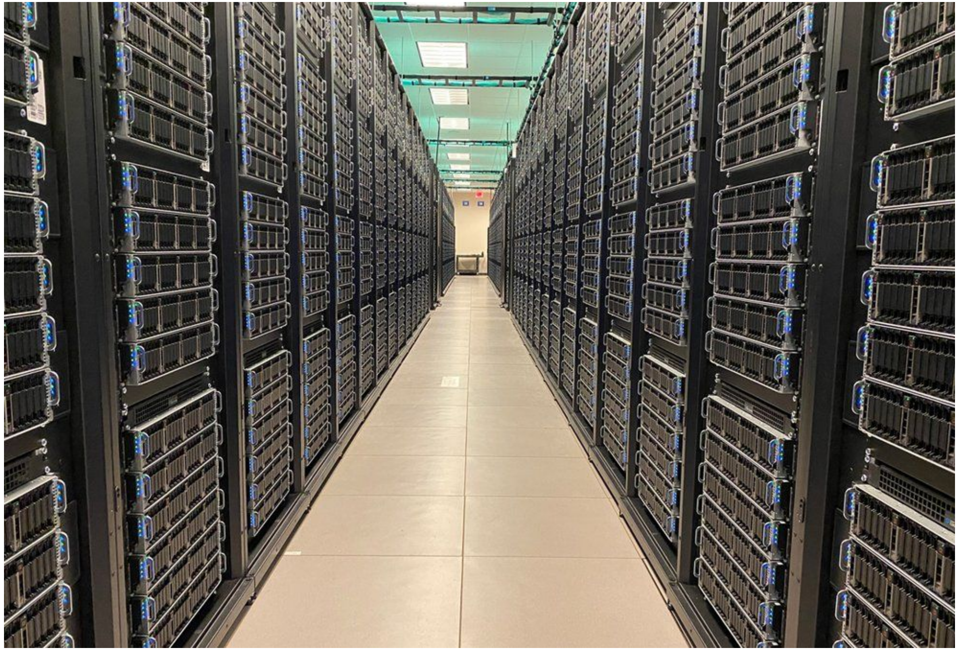
- Apache Spark distributed computing engine:
  - task dispatching
  - scheduling
  - communication over compute cluster
- Spark cluster managers requiring dedicated clusters:
  - Hadoop YARN
  - Apache MESOS
  - Kubernetes
- Rootless execution context challenges:
  - software dependencies hard to keep over HPC environment
  - no guarantee that same software list is available on each cluster
  - restricted service deployment in operating system user space

Frontera HPC Cluster

BIG HPC

The experience takes place over Frontera primary computing system:

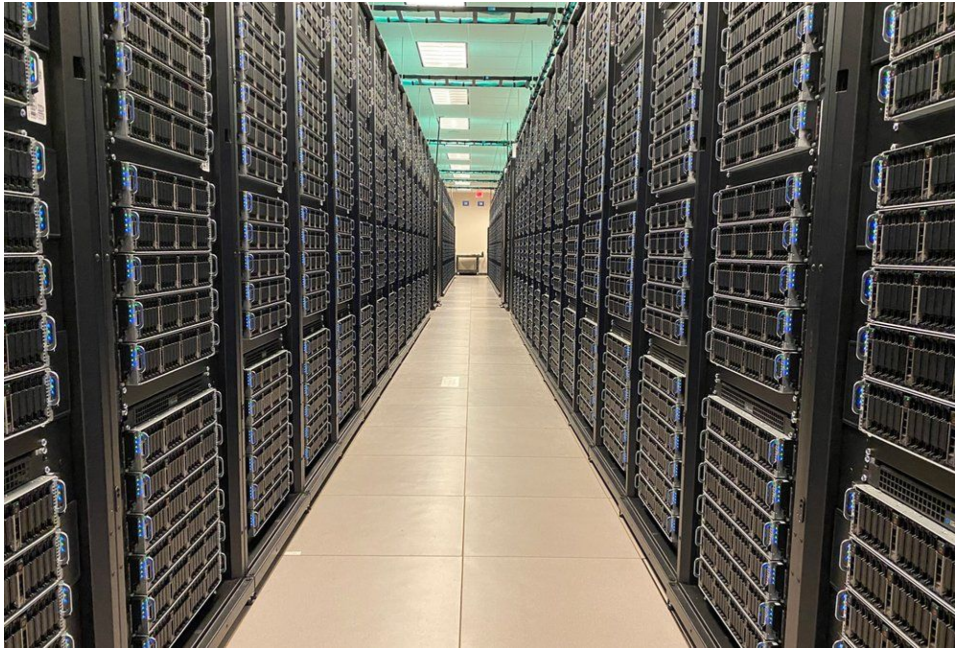
- Provided by Dell EMC with 8008 nodes
- Intel Xeon Platinum 8280 “Cascade Lake”
- 56 cores/node - 28 cores/socket
- Clock rate 2.7Ghz (Base Frequency)
- Peak Node Performance 4.8TF, Double Precision
- Memory/Node 192GB DDR-4
- Local Disk with 480GB SSD drive
- Mellanox Infiniband, HDR-100, capable of 200Gbps



Frontera HPC Cluster

The experience takes place over Frontera primary computing system:

- LINPACK benchmark: 23.5PF
- Theoretical peak performance: 38.7PF
- Storage composed by LUSTRE distributed file system:
  - based storage of 60PB
  - fast flash storage of 3PB
- LUSTRE OSTs (object storage targets):
  - based storage: home, work and scratch
  - flash storage: flash
- home: code and collected logs
- scratch: application workspace

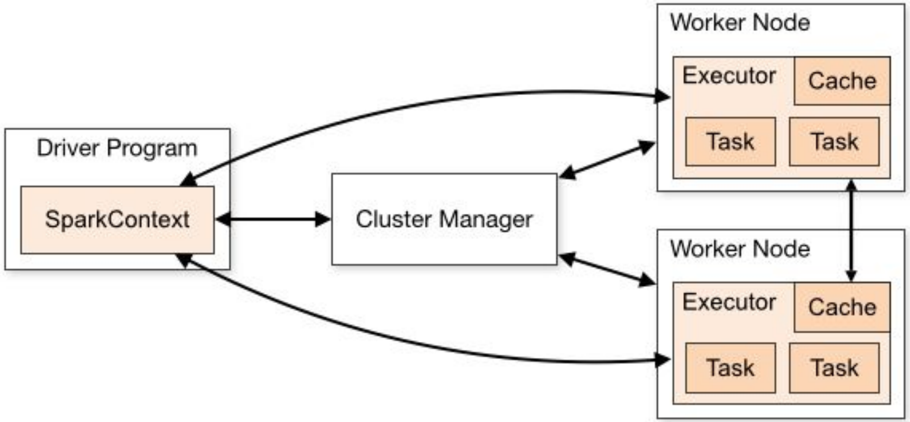


Apache Spark

- Analytics engine for large-scale data processing
- High level API in multiple languages (our example uses python)
- MLlib provide tools for machine learning
- RDD: resilient distributed dataset
  - working set for distributed programs over distributed shared memory
  - bypass limitations in the MapReduce cluster computing paradigm such as forced linear dataflow structure
- Processing workflow managed as a directed acyclic graph (DAG)
  - nodes are the RDDs
  - edges are the applied operations over the RDDs
- Handle acyclic and cyclic graphs that allows representation of iterative methods
- DAG iterative methods representation bypass another MapReduce limitation, providing a solution to express iterative algorithms

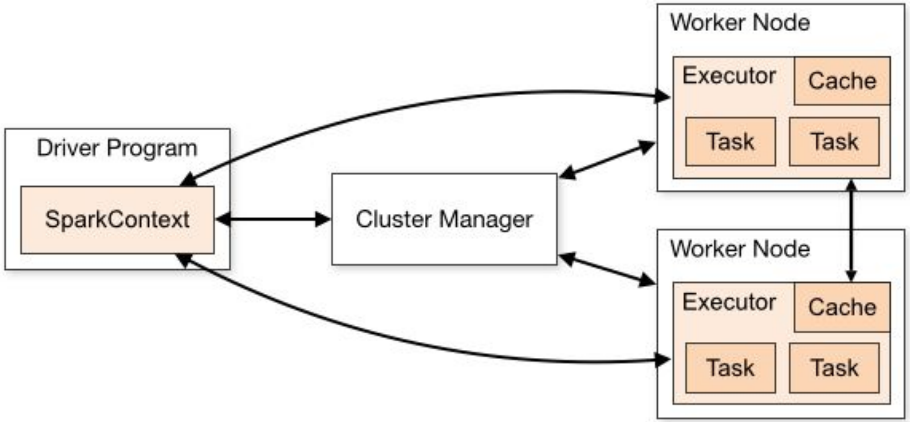
Apache Spark

- Engine deployment requires
  - a cluster manager
  - distributed storage system
- Cluster Manager: standalone Spark cluster
  - allows better control over the cluster operation
  - bypass rootless environment limitations
  - allocate resources across applications
  - send application code to the executors
- SparkContext: Spark driver program
  - responsible to send application code and its tasks to the executors
- Executors: spawn on the nodes
  - run the computations
  - manage the application data



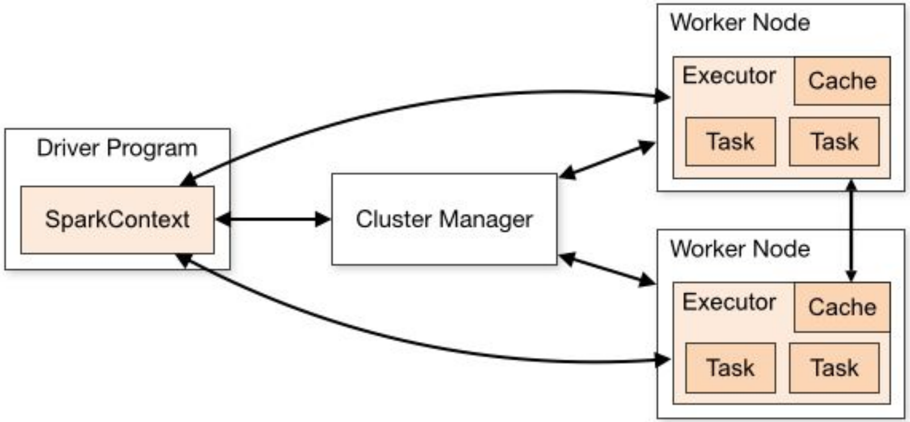
Apache Spark

- Block Manager:
  - application data cache
  - key-value store for data blocks
  - acts as a local cache for the driver and executors on every node
  - provides interface for upload and fetch blocks locally and remotely using memory, disk and external block stores
- Client execution modes:
  - cluster mode: driver process is launched on a worker node
  - client mode: driver remains on the client node that submitted the application
  - local mode: all processes of the application run on a single machine



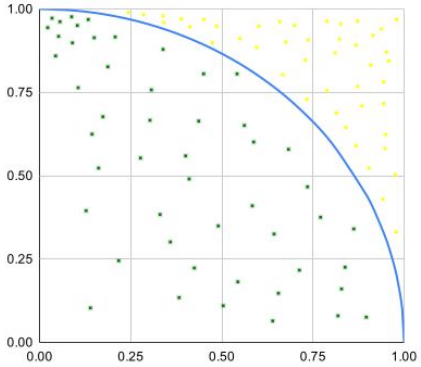
Apache Spark

- Local execution mode is used in our base tests with a single node
- With implemented Cluster Manager support scripts for multi-node tests it is possible to achieve cluster mode, bypassing client mode limitation of standalone deployment
- Support scripts are developed using bash command language:
    - manage deployment jobs
    - always available in any site
    - lesser maintenance effort since command language assure regression testing for any release
- Use apptainer to deploy the required software



Benchmark with Pi estimation

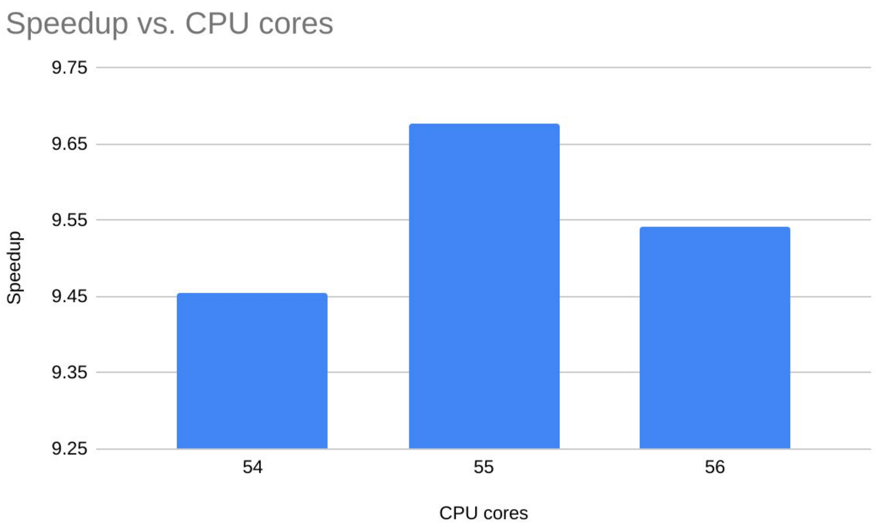
- Pi value estimation with Monte Carlo sampling methods
  - direct sampling
  - importance sampling
  - rejection sampling
- Best fit method for Pi value estimation is rejection sampling
  - allows to select samples within a region of the sampled distribution
  - simulate random points in a 2-D plane
  - domain is a square around a circle
  - square size equal the diameter of the circle
  - random number of points are generated inside the square
  - estimation is the ratio between points inside the circle and total number of generated points



Performance Optimization

Single Node

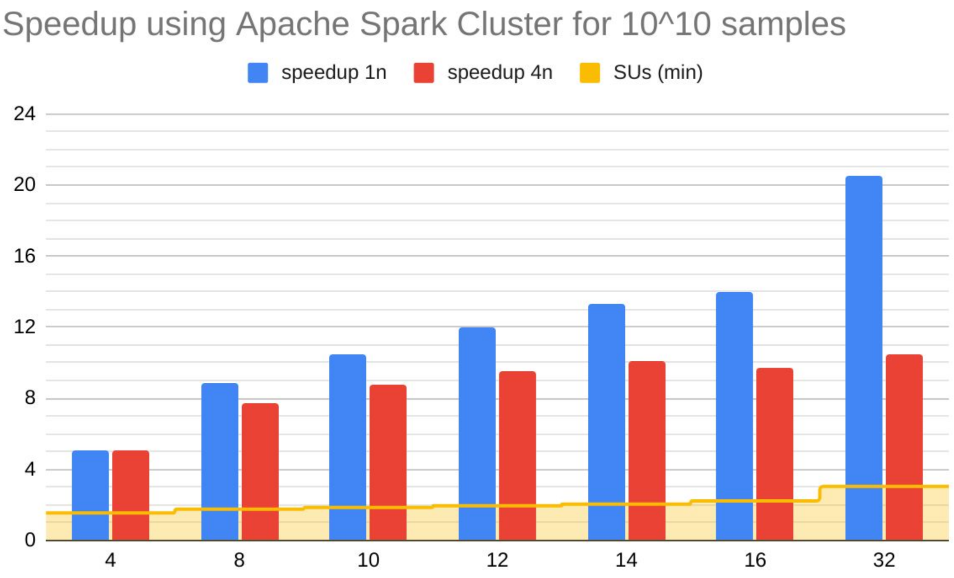
- Base test of 10^10 samples
- This test uses all resources of a single node
- Algorithm tends to be more CPU intensive
- Best speedup when leaving one core available for the operating system and driver for this environment



Performance Optimization

Executors multi-node scaling

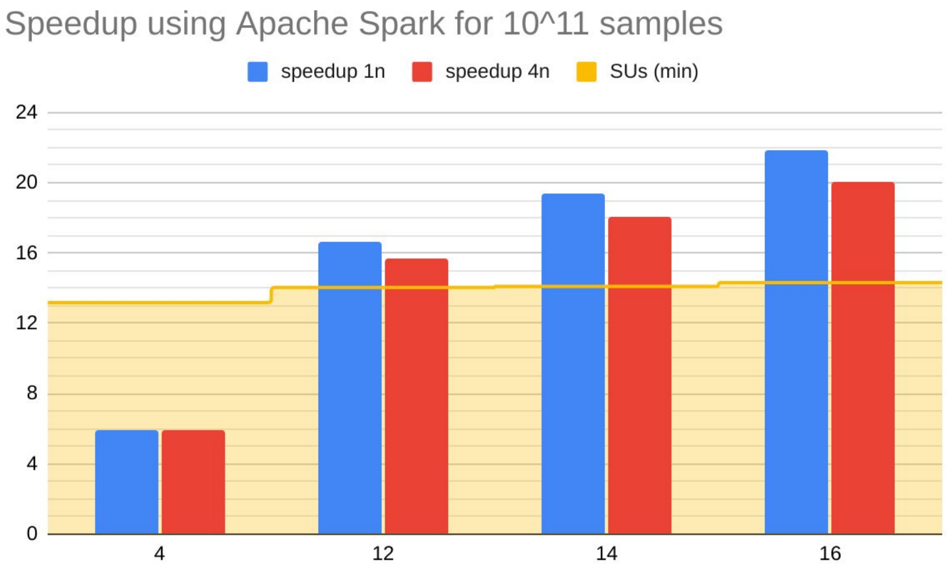
- We only use 140GB (at least 2.5GB per task) to keep executor memory equal in all nodes
- Start to scale out the single node test sample to different number of nodes
- We noted the superscalar effect of Intel processor caches that brought great speedups



Performance Optimization

Executors multi-node scaling

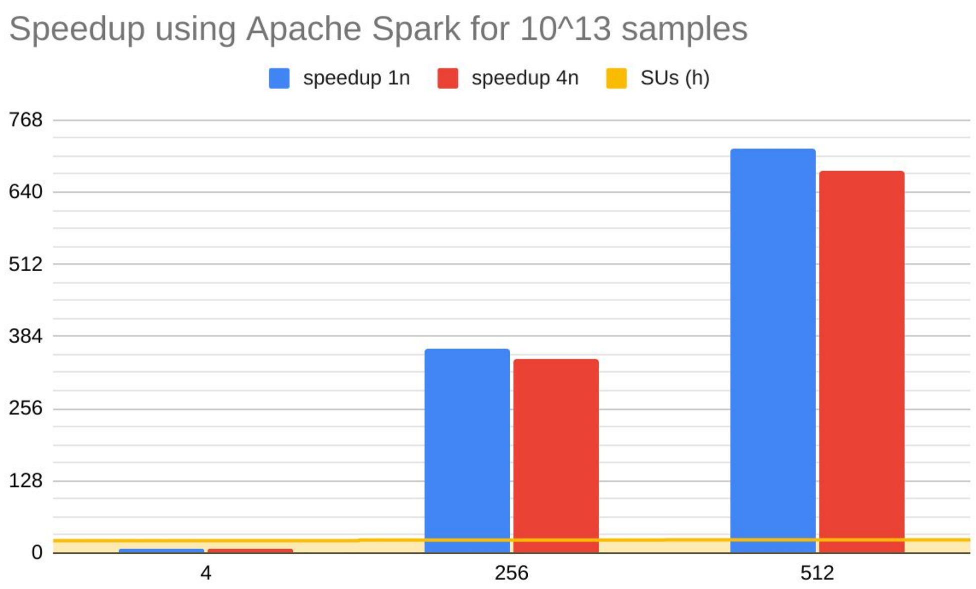
- Next step scaled up 10 times the sample
- This can still run over the 4 nodes available memory
- The performance with this sample have better results now with more nodes
- Communication takes lesser overhead with applied optimizations



Performance Optimization

Executors multi-node scaling

- Next step scaled up 1000 times the initial sample
- To run this simulation we need at least 256 nodes because of required memory size
- Here the superscalar effect overwhelms the previous results, turning evident the Frontera execution cost optimization for this job sizes



Thanks

- BigHPC consortium for supporting my research work on this case study
- To TACC team a special thanks with the research work around Frontera
- To LIP that is providing in Portugal the access to another HPC cluster to continue this research work and validate the optimizations with different hardware

BIG HPC HIGH PERFORMANCE COMPUTING

Partners:

INESCTEC

LIP

TEXAS The University of Texas at Austin

MACC Minho Advanced Computing Center

TACC

wavecom

Funding:

Cofinanciado por: COMPETE 2020

Lisb@20[20]

PORTUGAL 2020

UNIÃO EUROPEIA Fundo Europeu de Desenvolvimento Regional

FCT Fundação para a Ciência e a Tecnologia

UTAustin Portugal