Alchemist

An Apache Spark ⇔ MPI Interface

Mike Ringenburg, Cray, Inc

In collaboration with UC Berkeley riselab UC Berkeley

Legal Disclaimer

Information in this document is provided in connection with Cray Inc. products. No license, express or implied, to any intellectual property rights is granted by this document.

Cray Inc. may make changes to specifications and product descriptions at any time, without notice.

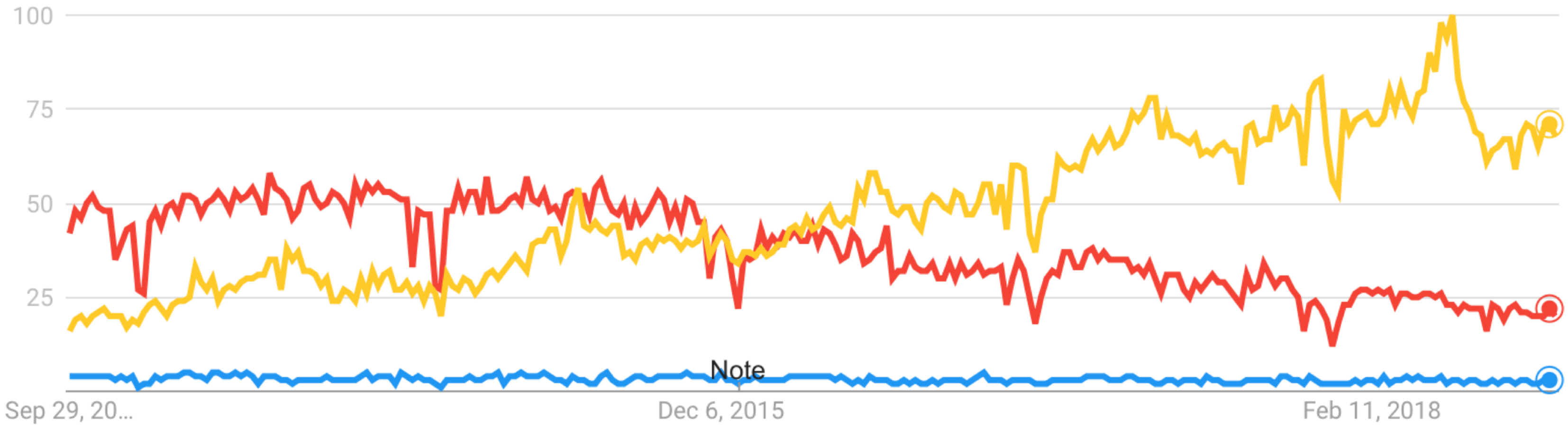
All products, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

Cray hardware and software products may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Cray uses codenames internally to identify products that are in development and not yet publically announced for release. Customers and other third parties are not authorized by Cray Inc. to use codenames in advertising, promotion or marketing and any use of Cray Inc. internal codenames is at the sole risk of the user.

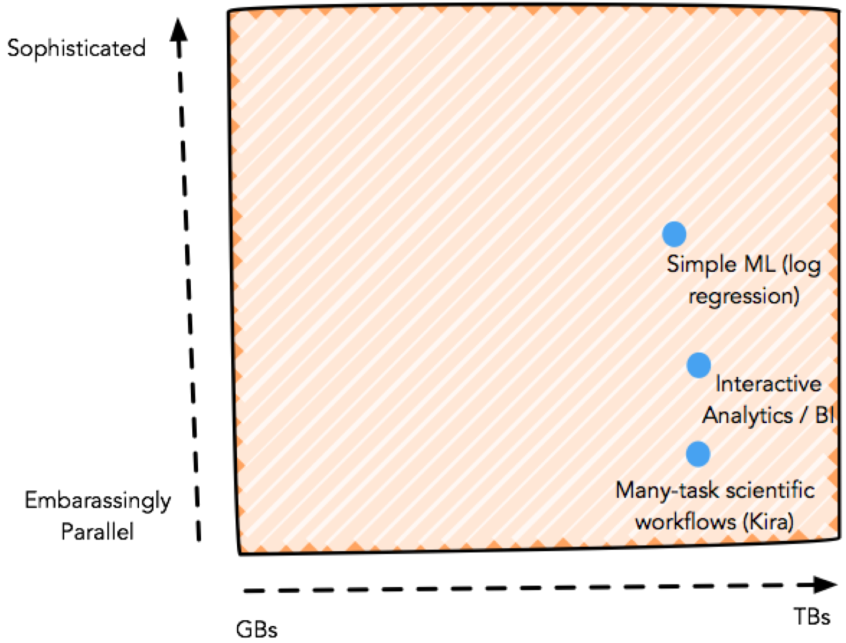
Performance tests and ratings are measured using specific systems and/or components and reflect the approximate performance of Cray Inc. products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.

Why should MPI codes interface with Spark?



Google trends popularity: MPI vs Hadoop vs Spark

Spark Use Cases



Q: What about less embarrassingly parallel computations?
A: Use Spark and MPI

Example: linear algebra in Spark

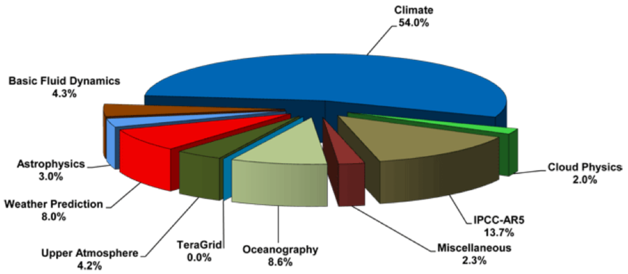
Pros for Spark:

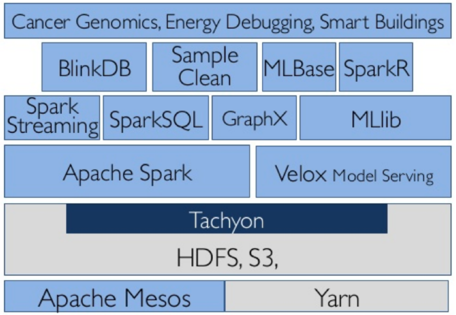
- Faster development, easier reuse
- One abstract uniform interface (RDD)
- An entire ecosystem that can be used before and after the NLA computations
- Spark can take advantage of available local linear algebra codes
- Automatic fault-tolerance, out-of-core support

Pros for MPI: Classical MPI-based linear algebra implementations will be faster and more efficient

Motivation

- **NERSC**: Spark for data-centric workloads and scientific analytics
- **RISELab**: characterization of linear algebra in Spark (MLlib, MLMatrix)
- **Cray**: users asking for Spark; understand performance concerns





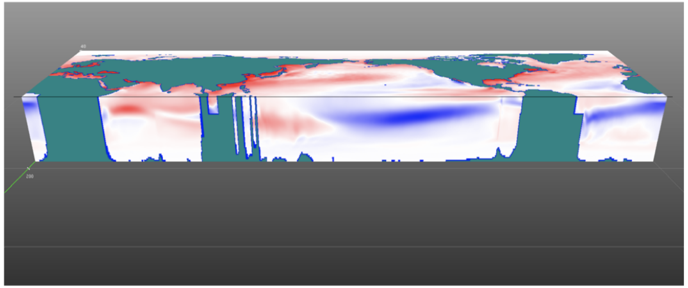
Case Study: Spark vs. MPI

- **Numerical linear algebra (NLA)** using Spark vs. MPI
- Computations performed on NERSC supercomputer **Cori** Phase 1, a Cray XC40
  - 2,388 compute nodes
  - 128 GB RAM/node, 32 2.3GHz Haswell cores/node
  - Lustre storage system, Cray Aries interconnect

A. Gittens et al. "Matrix factorizations at scale: A comparison of scientific data analytics in Spark and C+MPI using three case studies", 2016 IEEE International Conference on Big Data (Big Data), pages 204–213, Dec 2016.

Case Study: Spark vs. MPI

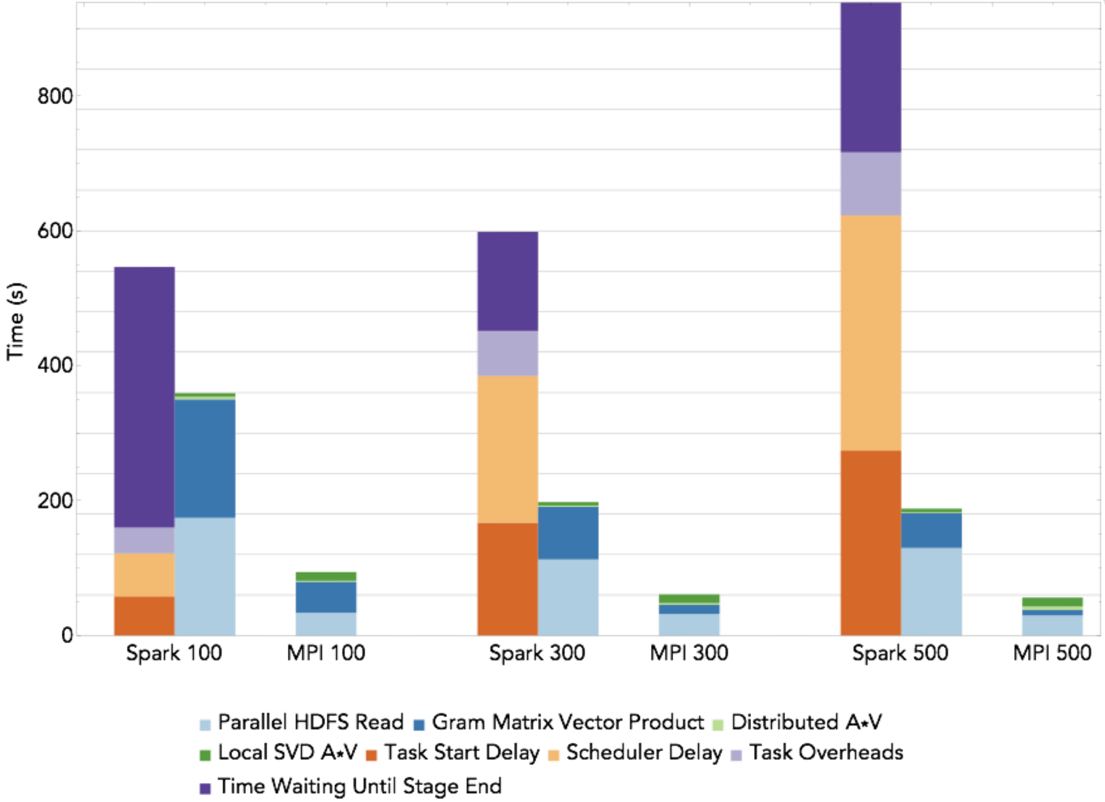
- **Numerical linear algebra (NLA)** using Spark vs. MPI
- Matrix factorizations considered include *truncated Singular Value Decomposition (SVD)*
- Data sets include
  - Ocean temperature data - 2.2 TB
  - Atmospheric data - 16 TB



A. Gittens et al. “Matrix factorizations at scale: A comparison of scientific data analytics in Spark and C+MPI using three case studies”, 2016 IEEE International Conference on Big Data (Big Data), pages 204–213, Dec 2016.

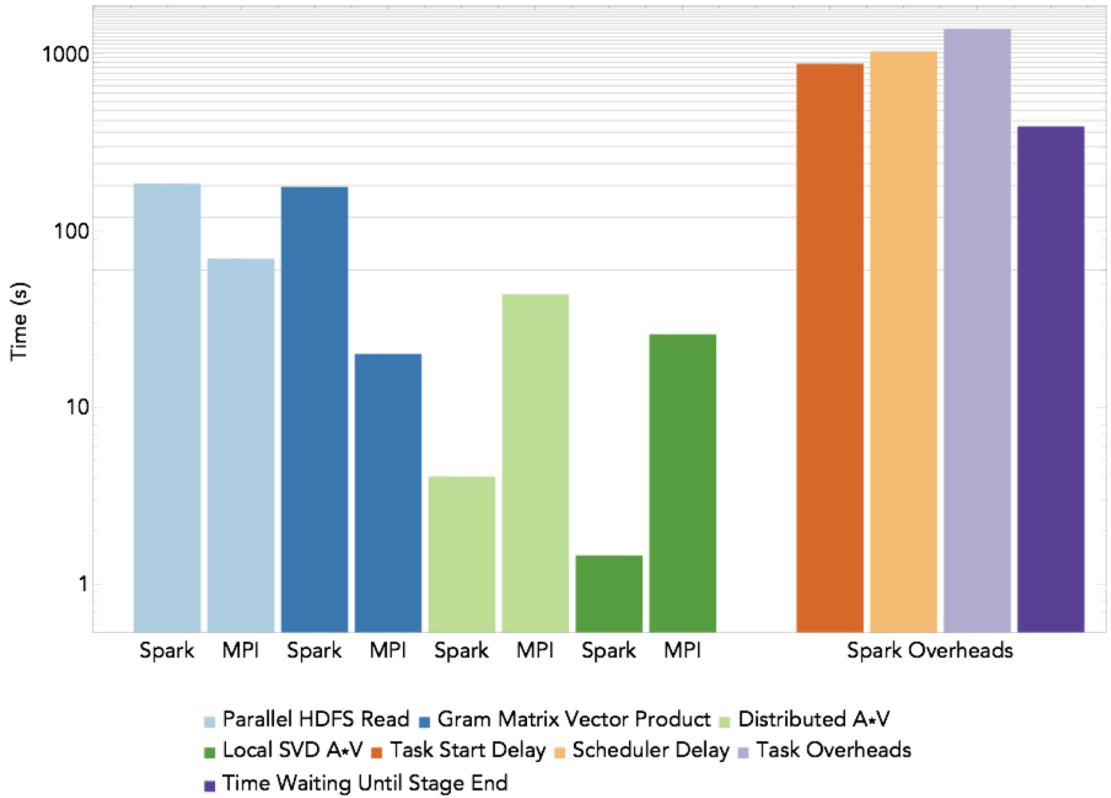
Case Study: Spark vs. MPI

Rank 20 SVD of 2.2TB ocean temperature data



Case Study: Spark vs. MPI

Rank 20 SVD of 16TB atmospheric data using 48K+ cores



Case Study: Spark vs. MPI

Lessons learned:

- With favorable data (tall and skinny) and well-adapted algorithms, linear algebra in Spark is 2x-26x slower than MPI when I/O is included
- Spark's overheads:
  - Can be order of magnitude higher than the actual computation times
  - Anti-scale
- **The gaps in performance suggest it may be better to interface with MPI-based codes from Spark**

- **Alchemist** interfaces between Apache Spark and *existing* or *custom* MPI-based libraries for **large-scale** linear algebra, machine learning, *etc.*
- ***Idea***:
  - Use Spark for regular data analysis workflow
  - When computationally intensive calculations are required, call relevant MPI-based codes from Spark using Alchemist, send results to Spark
- **Combine *high productivity* of Spark with *high performance* of MPI**

Alchemist

Target users:

- ***Scientific community:*** Use Spark for analysis of large scientific datasets by calling existing MPI-based libraries where appropriate
- ***Machine learning practitioners*** and ***data analysts***:
  - Better performance of a wide range of large-scale, computationally intensive ML and data analysis algorithms
  - For instance, SVD for principal component analysis, recommender systems, leverage scores, etc.

Basic Framework

- **Alchemist**: Acts as bridge between Spark and MPI-based libraries
- **Alchemist-Client Interface:** API for user, communicates with Alchemist via TCP/IP sockets
- **Alchemist-Library Interface:** Shared object, imports MPI library, provides generic interface for Alchemist to communicate with library

Basic Framework

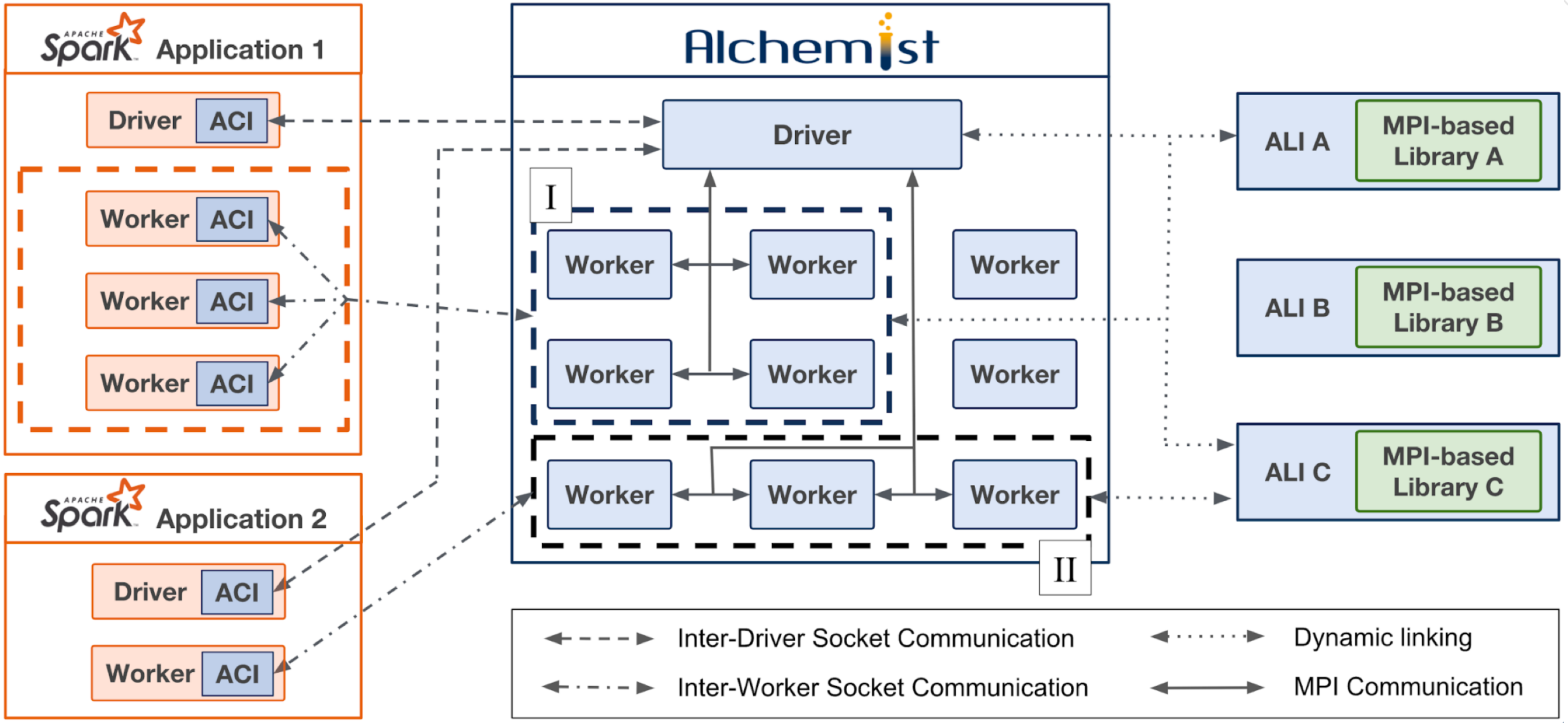
Basic workflow:

- Spark application:
  - Sends distributed dataset from RDD (`IndexedRowMatrix`) to Alchemist
  - Tells Alchemist what MPI-based code should be called
- Alchemist loads relevant MPI-based library, calls function, sends results to Spark

Basic Framework

- Alchemist can also load data from file
- Alchemist needs to store distributed data in appropriate format that can be used by MPI-based libraries:
    - Candidates: ***ScaLAPACK***, ***Elemental***, ***PLAPACK***
    - Alchemist currently uses Elemental, support for ScaLAPACK under development

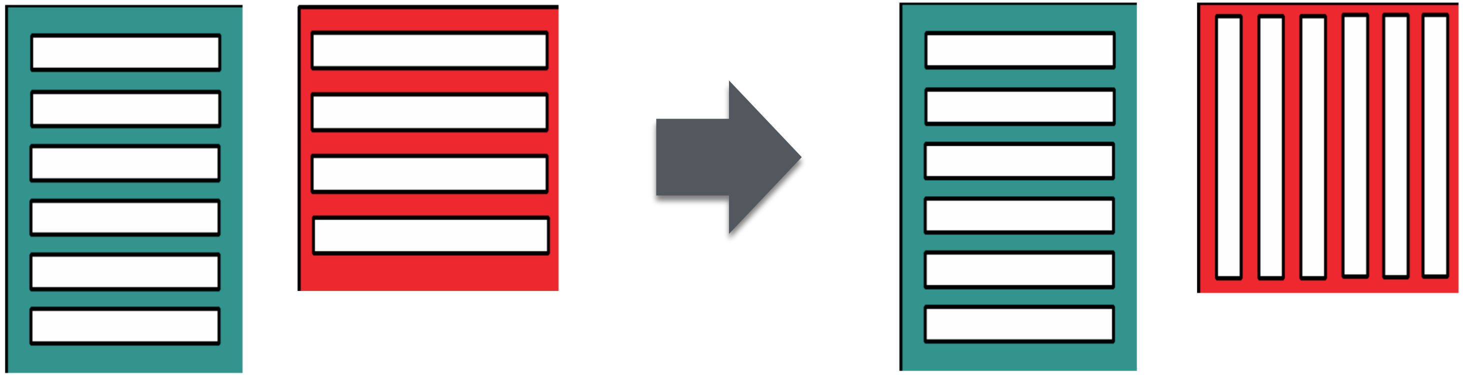
Alchemist Architecture



Sample API

```
import alchemist.{Alchemist, AlMatrix}
import alchemist.libA.QRDecomposition       // libA is sample MPI lib

// other code here ...

// sc is instance of SparkContext
val ac = new Alchemist.AlchemistContext(sc, numWorkers)
ac.registerLibrary("libA", ALIlibALocation)

// maybe other code here ...

val alA = AlMatrix(A)                        // A is IndexedRowMatrix

// routine returns QR factors of A as AlMatrix objects
val (alQ, alR) = QRDecomposition(alA)

// send data from Alchemist to Spark once ready
val Q = alQ.toIndexedRowMatrix()             // convert AlMatrix alQ to RDD
val R = alR.toIndexedRowMatrix()             // convert AlMatrix alR to RDD

// maybe other code here ...

ac.stop()                                    // release resources once no longer required
```

Example: Matrix Multiplication



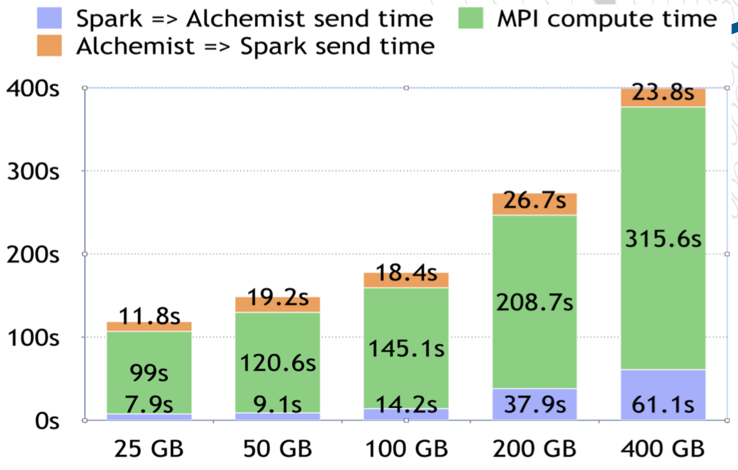
- Requires expensive shuffles in Spark, which is impractical:
  - Matrices/RDDs are row-partitioned
  - One matrix must be converted to be column-partitioned
  - *Requires an all-to-all shuffle that often fails* once the matrix is distributed

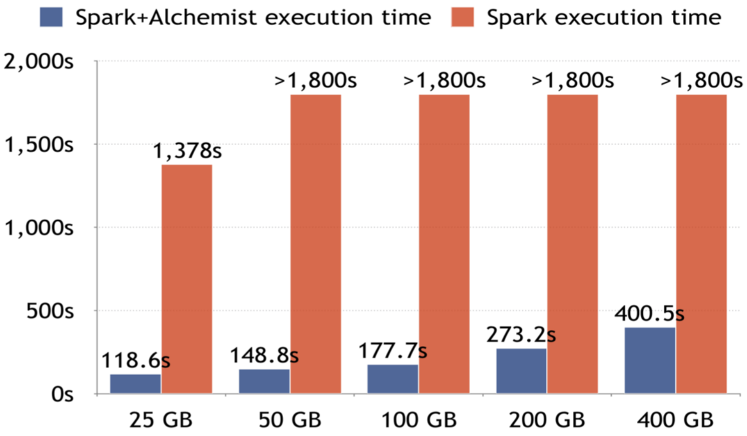
Example: Truncated SVD

Use Alchemist and MLlib to get rank 20 truncated SVD

Experiment Setup

- Spark: 22 nodes; Alchemist: 8 nodes
- A: m-by-10K, where m = 5M, 2.5M, 1.25M, 625K, 312.5K
- Ran jobs for at most 30 minutes (1800 s)

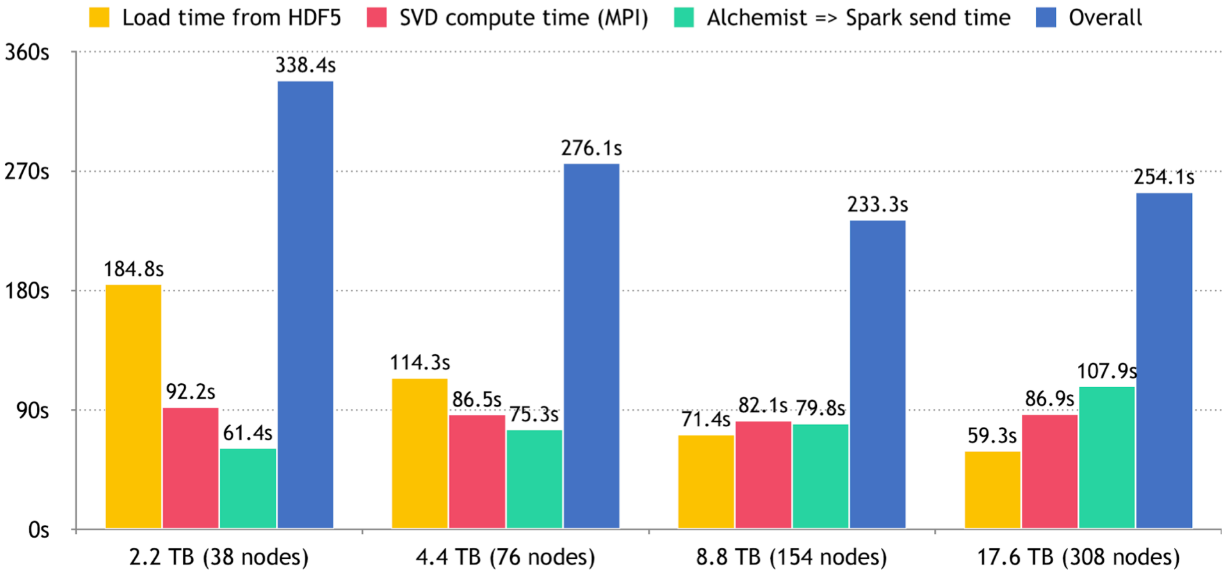




Example: Truncated SVD

Experiment Setup

- 2.2TB (6,177,583-by-46,752) ocean temperature data read in from HDF5 file
- Data replicated column-wise



Upcoming Features

- **PySpark, SparkR ⇔ MPI Interface**
    - Interface for Python => PySpark support
    - Future work: Interface for R
- **Direct Python interface, potential Dask integration**
- **More Functionality**
    - Support for sparse matrices
    - Support for MPI-based libraries built on ScaLAPACK
- **Alchemist and Containers**
    - Alchemist running in Docker and Kubernetes
    - Will enable Alchemist on clusters and the cloud

Limitations and Constraints

- **Two copies of data in memory**
- **Data transfer overhead** between Spark and Alchemist when data on different nodes
  - Subject to network disruptions and overload
- **MPI is not fault tolerant or elastic**
- **Lack of MPI-based libraries for machine learning**
  - No equivalent to MLlib currently available, closest is MaTEx
- Currently, **need to run Alchemist and Spark on separate nodes** -> more data transfer over interconnects -> larger overheads

Future Work

- **Apache Spark ⇔ X Interface**
  - Interest in connecting Spark with other libraries for distributed computing (e.g. Chapel)
- **Reduce communication costs**
  - Exploit locality
  - Reduce number of messages
  - Use communication protocols designed for underlying network infrastructure
- **Run as network service**
- **MPI computations with (basic) fault tolerance and elasticity**

github.com/project-alchemist/

References

- A. Gittens, K. Rothauge, M. W. Mahoney, *et al.*, "Alchemist: Accelerating Large-Scale Data Analysis by offloading to High-Performance Computing Libraries", 2018, *Proceedings of the 24th ACM SIGKDD International Conference,* Aug 2018, to appear
- A. Gittens, K. Rothauge, M. W. Mahoney, *et al.*, "Alchemist: An Apache Spark ⇔ MPI Interface", 2018, to appear in *CCPE Special Issue on Cray User Group Conference 2018*

Thanks to Cray Inc., DARPA and NSF for financial support