Convergence of High-Performance Computing and Big Data Analytics: Applications and Open Challenges

Silvina Caíno-Lores, PhD
scainolo@utk.edu
Acknowledgements

• **Argonne National Laboratory** (Bogdan Nicolae, Tom Peterka, Orçun Yildiz)
• **Cornell University** (Michel Cuendet, Ekaterina Kots, Harel Weinstein)
• **IBM** (Global University Program)
• **University Carlos III of Madrid** (Jesús Carretero)
• **University of Neuchatel** (Peter Kropf, Andrei Lapin)
• **University of New Mexico** (Hector Alexis Carrillo Cabada, Trilce Estrada)
• **University of Southern California** (Ewa Deelman, Rafael Ferreira da Silva, Tu Mai Anh Do, Loïc Pottier)
• **University of Tennessee-Knoxville** (Ian Lumdsen, Michela Taufer, Michael Wyatt)

And many more!
Outline

1. Motivation
2. Background
   - Big Data Analytics Ecosystem
   - High-Performance Computing Ecosystem
3. Convergence Challenges and Research Trends
4. Scientific Case Studies
   - Massively Parallel Analysis of Molecular Dynamics Simulations
   - Geographically Distributed Water Forecasting
5. Summary
Motivation
What is Large-Scale Computing?

Image credit to http://archive.vn/fJ0bX
Advancements in material technology, systems architecture, computational theory, and computation methods allowed the development of increasingly complex computing machines.
Large-scale computing is an instrumental component of science and technology development, as it enables the exploration of problems that cannot be addressed with personal computers.
Duality in Large-Scale Computing

High-performance computing (HPC) aggregates computing power to deliver maximum performance for a single, complex task

Big Data analytics (BDA) examines and exploits Big Data through business intelligence techniques for knowledge discovery in many-task settings

Hard and easy are mutually formed,
Long and short shape each other [...] 

*Tao Te Ching*, Chapter 2, Lao-Tze
# Duality in Large-Scale Computing

<table>
<thead>
<tr>
<th></th>
<th>HPC</th>
<th>BDA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Application</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Infrastructure</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Runtime</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Duality in Large-Scale Computing

<table>
<thead>
<tr>
<th></th>
<th>HPC</th>
<th>BDA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Application</strong></td>
<td>Compute-intensive</td>
<td>Data-intensive</td>
</tr>
<tr>
<td></td>
<td>Tightly-coupled</td>
<td>Loosely-coupled</td>
</tr>
<tr>
<td><strong>Infrastructure</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Runtime</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Duality in Large-Scale Computing

<table>
<thead>
<tr>
<th></th>
<th>HPC</th>
<th>BDA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Application</strong></td>
<td>Compute-intensive</td>
<td>Data-intensive</td>
</tr>
<tr>
<td></td>
<td>Tightly-coupled</td>
<td>Loosely-coupled</td>
</tr>
<tr>
<td><strong>Infrastructure</strong></td>
<td>Decoupled compute and storage</td>
<td>Co-located computation and data</td>
</tr>
<tr>
<td></td>
<td>High-speed interconnections</td>
<td>Elasticity is required</td>
</tr>
<tr>
<td><strong>Runtime</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Duality in Large-Scale Computing

<table>
<thead>
<tr>
<th></th>
<th>HPC</th>
<th>BDA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Application</strong></td>
<td>Compute-intensive</td>
<td>Data-intensive</td>
</tr>
<tr>
<td></td>
<td>Tightly-coupled</td>
<td>Loosely-coupled</td>
</tr>
<tr>
<td><strong>Infrastructure</strong></td>
<td>Decoupled compute and storage</td>
<td>Co-located computation and data</td>
</tr>
<tr>
<td></td>
<td>High-speed interconnections</td>
<td>Elasticity is required</td>
</tr>
<tr>
<td><strong>Runtime</strong></td>
<td><em>Process-centric</em></td>
<td><em>Data-centric</em></td>
</tr>
<tr>
<td></td>
<td>Intra- and inter-node parallelism</td>
<td>Batch and stream processing</td>
</tr>
</tbody>
</table>
The Need for Unity: Hybrid HPC-BDA Ecosystems
The Need for Unity: Hybrid HPC-BDA Ecosystems

Modern research areas rely on applications with mixed HPC and BDA requirements

- Data-intensive scientific computing
- High-performance data analytics (especially certain areas of ML)
The Need for Unity: Hybrid HPC-BDA Ecosystems

Modern research areas rely on applications with mixed HPC and BDA requirements
  ● Data-intensive scientific computing
  ● High-performance data analytics (especially certain areas of ML)

Large-scale infrastructures are merging into decentralised, heterogeneous architectures
  ● Deeper storage hierarchy
  ● Heterogeneous and high-performance cloud
The Need for Unity: Hybrid HPC-BDA Ecosystems

Modern research areas rely on applications with mixed HPC and BDA requirements
- Data-intensive scientific computing
- High-performance data analytics (especially certain areas of ML)

Large-scale infrastructures are merging into decentralised, heterogeneous architectures
- Deeper storage hierarchy
- Heterogeneous and high-performance cloud

New computing paradigms
- IoT (smart cities, surveillance, autonomous vehicles)
- Edge with supercomputing support
The Need for Unity: Hybrid HPC-BDA Ecosystems

Modern research areas rely on applications with mixed HPC and BDA requirements
- Data-intensive scientific computing
- High-performance data analytics (especially certain areas of ML)

Large-scale infrastructures are merging into decentralised, heterogeneous architectures
- Deeper storage hierarchy
- Heterogeneous and high-performance cloud

New computing paradigms
- IoT (smart cities, surveillance, autonomous vehicles)
- Edge with supercomputing support

How can we converge HPC and BDA ecosystems without losing their respective benefits?
Background
HPC Ecosystem

High-performance computing (HPC) aggregates computing power to deliver maximum performance for a single, complex task.

Tightly related to the concept of supercomputing:

- Pushes HPC to the highest operational rate of the available technology
- Involves massive investment in hardware development, runtime design, and daily operational costs
HPC Ecosystem

High-performance computing (HPC) aggregates computing power to deliver maximum performance for a single, complex task

- Reserved for strategic sectors that rely on complex numerical applications that cannot be run on commodity machines
  - Aviation, energy, pharmaceutical, oil and gas, and automotive, high-end scientific research on climate, medicine, bioinformatics, and physics.
- Focused on parallel processing on many cores and nodes
- Dependant on network-intensive data transfers between compute and storage nodes
HPC Ecosystem
HPC Ecosystem: Infrastructure

- Storage and computation are not located in the same nodes
- Networks are isolated to avoid the interference of I/O operations with computation communications
- New infrastructure architectures incorporate deeper memory hierarchies and local storage
- Accelerators like GPGPUs, FPGAs and TPUs became a standard in modern supercomputers
HPC Ecosystem: Software Stack

- Applications aim to run at the maximum level of parallelism provided by supercomputers in order to reduce execution time and increase scalability.

Big Data analytics (BDA) examines and exploits Big Data through business intelligence techniques for knowledge discovery in many-task settings.

- Big Data is not:
  - Just a very large dataset
  - A computing model (e.g. Cloud Computing)
  - A collection of techniques (e.g. Data science, Data analytics)

- The multi-V model defines Big Data by capturing the challenges of working with modern datasets:
  - **Volume**: necessary in order to get insight from analytics tools, but challenging!
  - **Velocity**: batch processing, streaming, near- and real-time speed
  - **Variety**: data can be highly heterogeneous and may be unstructured
  - And more as needed: **Veracity, Value, Variability**, etc.
BDA Ecosystem

**Big Data analytics (BDA)** examines and exploits Big Data through **business intelligence techniques** for knowledge discovery in many-task settings.

- There are many ways to explore the nature and relationships between datasets
  - Data mining
  - Statistical analysis
  - Data visualisation
  - Artificial intelligence
  - Natural language processing

- Some of these techniques have been around for years and they have been revamped due to their good adaptability to very large data sets with minimal data preparation.
Big Data analytics (BDA) examines and exploits Big Data through business intelligence techniques for knowledge discovery in many-task settings.
Big Data analytics (BDA) examines and exploits Big Data through business intelligence techniques for knowledge discovery in many-task settings.

- Big Data are inherently massively parallel, and typically task-based.
- Require support from specialised platforms and infrastructures:
  - Map-Reduce
  - In-database analytics
  - In-memory databases
  - Columnar data stores
  - Cloud computing at all levels of abstractions (from IaaS to FaaS)
  - Edge and Fog support
Upcoming scenarios might provide terabytes of data per hour
Centralising all data is no longer viable, especially if low latency is expected
Hierarchical infrastructures support efficient real-time operations for monitoring and decision making
BDA Ecosystem: Software Stack

- Programming models provide a data processing layer able to abstract resource allocation, data management and task execution.
- Minimising data movements is very important for the final performance.
- More modern in-memory approaches have enhanced performance and introduced more expressive programming models.

Convergence Challenges and Research Trends
Analysis of HPC Ecosystem

Pros

- Exploit maximum parallelism
- Low overhead
- Generalist interface
- Bare-metal access
- Top-tier hardware including accelerators
- Centralised
- Fast interconnections

Cons

- Limited data abstractions
- Steep learning curve
- No native provenance nor replication
- Low portability
- Decoupled storage reduces locality
- Limited availability
## Analysis of HPC Ecosystem

<table>
<thead>
<tr>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parallel programming platforms</strong></td>
<td></td>
</tr>
<tr>
<td>● Exploit maximum parallelism</td>
<td>● Limited data abstractions</td>
</tr>
<tr>
<td>● Low overhead</td>
<td>● Steep learning curve</td>
</tr>
<tr>
<td>● Generalist interface</td>
<td>● No native provenance nor replication</td>
</tr>
<tr>
<td>● Bare-metal access</td>
<td>● Low portability</td>
</tr>
<tr>
<td>● Top-tier hardware including accelerators</td>
<td>● Decoupled storage reduces locality</td>
</tr>
<tr>
<td></td>
<td>● Limited availability</td>
</tr>
</tbody>
</table>
Analysis of HPC Ecosystem

<table>
<thead>
<tr>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parallel programming platforms</strong></td>
<td></td>
</tr>
</tbody>
</table>
| ● Exploit maximum parallelism  
● Low overhead  
● Generalist interface  
● Bare-metal access |  |

- Limited data abstractions  
- Steep learning curve  
- No native provenance nor replication  
- Low portability  
- Decoupled storage reduces locality  
- Limited availability
Analysis of HPC Ecosystem

<table>
<thead>
<tr>
<th>Parallel programming platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pros</td>
</tr>
<tr>
<td>● Exploit maximum parallelism</td>
</tr>
<tr>
<td>● Low overhead</td>
</tr>
<tr>
<td>● Generalist interface</td>
</tr>
<tr>
<td>● Bare-metal access</td>
</tr>
<tr>
<td>Cons</td>
</tr>
<tr>
<td>● Limited data abstractions</td>
</tr>
<tr>
<td>● Steep learning curve</td>
</tr>
<tr>
<td>● No native provenance nor replication</td>
</tr>
<tr>
<td>● Low portability</td>
</tr>
</tbody>
</table>
## Analysis of HPC Ecosystem

<table>
<thead>
<tr>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parallel programming platforms</strong></td>
<td><strong>Supercomputing infrastructures</strong></td>
</tr>
<tr>
<td>● Exploit maximum parallelism</td>
<td></td>
</tr>
<tr>
<td>● Low overhead</td>
<td></td>
</tr>
<tr>
<td>● Generalist interface</td>
<td></td>
</tr>
<tr>
<td>● Bare-metal access</td>
<td></td>
</tr>
<tr>
<td></td>
<td>● Limited data abstractions</td>
</tr>
<tr>
<td></td>
<td>● Steep learning curve</td>
</tr>
<tr>
<td></td>
<td>● No native provenance nor replication</td>
</tr>
<tr>
<td></td>
<td>● Low portability</td>
</tr>
</tbody>
</table>
## Analysis of HPC Ecosystem

<table>
<thead>
<tr>
<th></th>
<th>Parallel programming platforms</th>
<th>Supercomputing infrastructures</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Pros</strong></td>
<td>● Exploit maximum parallelism&lt;br&gt;● Low overhead&lt;br&gt;● Generalist interface&lt;br&gt;● Bare-metal access</td>
<td>● Top-tier hardware including accelerators&lt;br&gt;● Centralised&lt;br&gt;● Fast interconnections</td>
</tr>
<tr>
<td><strong>Cons</strong></td>
<td>● Limited data abstractions&lt;br&gt;● Steep learning curve&lt;br&gt;● No native provenance nor replication&lt;br&gt;● Low portability</td>
<td></td>
</tr>
</tbody>
</table>
## Analysis of HPC Ecosystem

<table>
<thead>
<tr>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parallel programming platforms</td>
<td>Supercomputing infrastructures</td>
</tr>
<tr>
<td>● Exploit maximum parallelism</td>
<td>● Top-tier hardware including accelerators</td>
</tr>
<tr>
<td>● Low overhead</td>
<td>● Centralised</td>
</tr>
<tr>
<td>● Generalist interface</td>
<td>● Fast interconnections</td>
</tr>
<tr>
<td>● Bare-metal access</td>
<td></td>
</tr>
</tbody>
</table>
## Analysis of HPC Ecosystem

<table>
<thead>
<tr>
<th></th>
<th>Parallel programming platforms</th>
<th>Supercomputing infrastructures</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Pros</strong></td>
<td>● Exploit maximum parallelism</td>
<td>● Top-tier hardware including accelerators</td>
</tr>
<tr>
<td></td>
<td>● Low overhead</td>
<td>● Centralised</td>
</tr>
<tr>
<td></td>
<td>● Generalist interface</td>
<td>● Fast interconnections</td>
</tr>
<tr>
<td></td>
<td>● Bare-metal access</td>
<td></td>
</tr>
<tr>
<td><strong>Cons</strong></td>
<td>● Limited data abstractions</td>
<td>● Decoupled storage reduces locality</td>
</tr>
<tr>
<td></td>
<td>● Steep learning curve</td>
<td>● Limited availability</td>
</tr>
<tr>
<td></td>
<td>● No native provenance nor replication</td>
<td></td>
</tr>
<tr>
<td></td>
<td>● Low portability</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Analysis of BDA Ecosystem

Pros

- Fault-tolerance by design
- Transparent data locality
- Productive programming interface
- Synergetic pre-built tools for composite jobs
- Flexibility through virtualisation
- Diverse local storage (NVRAM, SSD, scratch)
- Elasticity
- Massive geographic distribution

Cons

- Low resource management control
- Significant memory overhead
- Poor support of binary input
- Deep software and communication stack
- Poor integration with simulation kernels
- Resource sharing
- High latency
- Enterprise hardware
- Privacy concerns
## Analysis of BDA Ecosystem

<table>
<thead>
<tr>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data-centric platforms</strong></td>
<td><strong>Low resource management control</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Significant memory overhead</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Poor support of binary input</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Deep software and communication stack</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Poor integration with simulation kernels</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Resource sharing</strong></td>
</tr>
<tr>
<td></td>
<td><strong>High latency</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Enterprise hardware</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Privacy concerns</strong></td>
</tr>
</tbody>
</table>
## Analysis of BDA Ecosystem

<table>
<thead>
<tr>
<th>Data-centric platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>● Fault-tolerance by design</td>
</tr>
<tr>
<td>● Transparent data locality</td>
</tr>
<tr>
<td>● Productive programming interface</td>
</tr>
<tr>
<td>● Synergetic pre-built tools for composite jobs</td>
</tr>
</tbody>
</table>

### Pros

- Fault-tolerance by design
- Transparent data locality
- Productive programming interface
- Synergetic pre-built tools for composite jobs

### Cons

- Low resource management control
- Significant memory overhead
- Poor support of binary input
- Deep software and communication stack
- Poor integration with simulation kernels
- Resource sharing
- High latency
- Enterprise hardware
- Privacy concerns
## Analysis of BDA Ecosystem

<table>
<thead>
<tr>
<th>Data-centric platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Pros</strong></td>
</tr>
<tr>
<td>● Fault-tolerance by design</td>
</tr>
<tr>
<td>● Transparent data locality</td>
</tr>
<tr>
<td>● Productive programming interface</td>
</tr>
<tr>
<td>● Synergetic pre-built tools for composite jobs</td>
</tr>
<tr>
<td><strong>Cons</strong></td>
</tr>
<tr>
<td>● Low resource management control</td>
</tr>
<tr>
<td>● Significant memory overhead</td>
</tr>
<tr>
<td>● Poor support of binary input</td>
</tr>
<tr>
<td>● Deep software and communication stack</td>
</tr>
<tr>
<td>● Poor integration with simulation kernels</td>
</tr>
</tbody>
</table>
## Analysis of BDA Ecosystem

<table>
<thead>
<tr>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data-centric platforms</td>
<td>Cloud and Fog environments</td>
</tr>
<tr>
<td>- Fault-tolerance by design</td>
<td>- Low resource management control</td>
</tr>
<tr>
<td>- Transparent data locality</td>
<td>- Significant memory overhead</td>
</tr>
<tr>
<td>- Productive programming interface</td>
<td>- Poor support of binary input</td>
</tr>
<tr>
<td>- Synergetic pre-built tools for composite jobs</td>
<td>- Deep software and communication stack</td>
</tr>
<tr>
<td></td>
<td>- Poor integration with simulation kernels</td>
</tr>
</tbody>
</table>
## Analysis of BDA Ecosystem

<table>
<thead>
<tr>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data-centric platforms</strong></td>
<td><strong>Cloud and Fog environments</strong></td>
</tr>
<tr>
<td>• Fault-tolerance by design</td>
<td>• Flexibility through virtualisation</td>
</tr>
<tr>
<td>• Transparent data locality</td>
<td>• Diverse local storage (NVRAM, SSD,</td>
</tr>
<tr>
<td>• Productive programming interface</td>
<td>scratch)</td>
</tr>
<tr>
<td>• Synergetic pre-built tools for</td>
<td>• Elasticity</td>
</tr>
<tr>
<td>composite jobs</td>
<td>• Massive geographic distribution</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Cons</strong></td>
<td></td>
</tr>
<tr>
<td>• Low resource management control</td>
<td>• Poor integration with simulation</td>
</tr>
<tr>
<td>• Significant memory overhead</td>
<td>kernels</td>
</tr>
<tr>
<td>• Poor support of binary input</td>
<td></td>
</tr>
<tr>
<td>• Deep software and communication stack</td>
<td></td>
</tr>
<tr>
<td>• Poor integration with simulation</td>
<td></td>
</tr>
<tr>
<td>kernels</td>
<td></td>
</tr>
</tbody>
</table>
## Analysis of BDA Ecosystem

<table>
<thead>
<tr>
<th>Pros</th>
<th>Data-centric platforms</th>
<th>Cloud and Fog environments</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>● Fault-tolerance by design</td>
<td>● Flexibility through virtualisation</td>
</tr>
<tr>
<td></td>
<td>● Transparent data locality</td>
<td>● Diverse local storage (NVRAM, SSD, scratch)</td>
</tr>
<tr>
<td></td>
<td>● Productive programming interface</td>
<td>● Elasticity</td>
</tr>
<tr>
<td></td>
<td>● Synergetic pre-built tools for composite jobs</td>
<td>● Massive geographic distribution</td>
</tr>
<tr>
<td>Cons</td>
<td>● Low resource management control</td>
<td>● Resource sharing</td>
</tr>
<tr>
<td></td>
<td>● Significant memory overhead</td>
<td>● High latency</td>
</tr>
<tr>
<td></td>
<td>● Poor support of binary input</td>
<td>● Enterprise hardware</td>
</tr>
<tr>
<td></td>
<td>● Deep software and communication stack</td>
<td>● Privacy concerns</td>
</tr>
<tr>
<td></td>
<td>● Poor integration with simulation kernels</td>
<td></td>
</tr>
</tbody>
</table>
## Analysis of BDA Ecosystem

<table>
<thead>
<tr>
<th>Pros</th>
<th>Data-centric platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>● Fault-tolerance by design</td>
<td></td>
</tr>
<tr>
<td>● Transparent data locality</td>
<td></td>
</tr>
<tr>
<td>● Productive programming interface</td>
<td></td>
</tr>
<tr>
<td>● Synergetic pre-built tools for composite jobs</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cons</th>
<th>Cloud and Fog environments</th>
</tr>
</thead>
<tbody>
<tr>
<td>● Low resource management control</td>
<td></td>
</tr>
<tr>
<td>● Significant memory overhead</td>
<td></td>
</tr>
<tr>
<td>● Poor support of binary input</td>
<td></td>
</tr>
<tr>
<td>● Deep software and communication stack</td>
<td></td>
</tr>
<tr>
<td>● Poor integration with simulation kernels</td>
<td></td>
</tr>
<tr>
<td>● Flexibility through virtualisation</td>
<td></td>
</tr>
<tr>
<td>● Diverse local storage (NVRAM, SSD, scratch)</td>
<td></td>
</tr>
<tr>
<td>● Elasticity</td>
<td></td>
</tr>
<tr>
<td>● Massive geographic distribution</td>
<td></td>
</tr>
<tr>
<td>● Resource sharing</td>
<td></td>
</tr>
<tr>
<td>● High latency</td>
<td></td>
</tr>
<tr>
<td>● Enterprise hardware</td>
<td></td>
</tr>
<tr>
<td>● Privacy concerns</td>
<td></td>
</tr>
</tbody>
</table>
State of HPC-BDA Convergence
State of HPC-BDA Convergence

1. HPC enhances BDA
   ○ Implementation of data-centric models in parallel runtimes (e.g. MPI)
   ○ Adaptation to supercomputing facilities (high-speed interconnections, accelerators)
State of HPC-BDA Convergence

1. HPC enhances BDA
   ○ Implementation of data-centric models in parallel runtimes (e.g. MPI)
   ○ Adaptation to supercomputing facilities (high-speed interconnections, accelerators)

2. BDA enhances HPC
   ○ Adaptation of scientific problems to data-centric programming models
   ○ Domain-specific data-centric frameworks for visualisation and data management
   ○ Integration of advanced analytics techniques in scientific domains (e.g. ML)
State of HPC-BDA Convergence

1. **HPC enhances BDA**
   - Implementation of data-centric models in parallel runtimes (e.g. MPI)
   - Adaptation to supercomputing facilities (high-speed interconnections, accelerators)

2. **BDA enhances HPC**
   - Adaptation of scientific problems to data-centric programming models
   - Domain-specific data-centric frameworks for visualisation and data management
   - Integration of advanced analytics techniques in scientific domains (e.g. ML)

3. **True interoperability**
   - Development of programming interfaces (mainly Spark–MPI)
   - Lack of generality, flexibility, and theoretical foundation
Convergence Challenges
Convergence Challenges

- **Programming and data models**
  - Differences in cultures and tools (numeric data structures vs. flexible layouts)
  - Coexistence of stream, batch, and iterative models

- **Runtimes and platforms**
  - Common software ecosystem for application development
  - Interoperability between data formats and programming languages

- **Computing and storage infrastructures**
  - Divergence in underlying storage
  - Memory limitations
Convergence Challenges

● Programming and data models
  ○ Differences in cultures and tools (numeric data structures vs. flexible layouts)
  ○ Coexistence of stream, batch, and iterative models

● Runtimes and platforms
  ○ Common software ecosystem for application development
  ○ Interoperability between data formats and programming languages
Convergence Challenges

● **Programming and data models**
  ○ Differences in cultures and tools (numeric data structures vs. flexible layouts)
  ○ Coexistence of stream, batch, and iterative models

● **Runtimes and platforms**
  ○ Common software ecosystem for application development
  ○ Interoperability between data formats and programming languages

● **Computing and storage infrastructures**
  ○ Divergence in underlying storage
  ○ Memory limitations
Scientific Case Studies: Massively Parallel Analysis of Molecular Dynamics Simulations
Classical Molecular Dynamics (MD) Simulations

• A MD simulation comprises of hundreds of thousands of MD jobs
• Each job performs hundreds of thousands of MD steps

Molecular Dynamics Jobs

1. MD step computes **forces** on single atoms (e.g., bond, dihedrals, nonbond)
2. Forces are added to compute **acceleration**
3. Acceleration is used to update **velocities**
4. Velocities are used to update the **atom positions**
5. Every $N$ steps (stride)
   - **Store 3D snapshot or frame**
Molecular Dynamics Jobs

1. MD step computes **forces** on single atoms (e.g., bond, dihedrals, nonbond)
2. Forces are added to compute **acceleration**
3. Acceleration is used to update **velocities**
4. Velocities are used to update the **atom positions**
5. Every $N$ steps (stride)  
   - **Store 3D snapshot or frame**
Scientist-Driven Analysis of MD Trajectories

Frames (or snapshots) of an MD trajectory with a stride of 5 steps:

Frame 55 | Frame 60 | Frame 65 | Frame 70 | Frame 75 | Frame 80

---

Visualization

Scientist

---

Scientist-Driven Analysis of MD Trajectories

Simulation and analysis are isolated!

Visualization

Scientist


---

#!/bin/bash
#SBATCH --time=4:00:00
#SBATCH --nodes=1

---

Frames (or snapshots) of an MD trajectory with a stride of 5 steps:
An holistic approach that co-locates simulation and analysis can benefit from:

- Natural integration with data streaming
- Massive task parallelism
- In-memory storage

However, we must find a way to integrate MD simulations with the analytics:

**Step 1**
Leverage the *in situ* approach for data analytics

**Step 2**
Introduce support for the orchestration of multiple tasks
From HPC to BDA MD Simulations

An holistic approach that co-locates simulation and analysis can benefit from

- Natural integration with data streaming
- Massive task parallelism
- In-memory storage

However, we must find a way to integrate MD simulations with the analytics

Step 1
Leverage the *in situ* approach for data analytics

Step 2
Introduce support for the orchestration of multiple tasks
In Situ Analysis of MD Trajectories

Frames (or snapshots) of an MD trajectory with a stride of 5 steps:

Collective variables 
(t = 55)

Collective variables 
(t = 60)

Collective variables 
(t = 65)

Collective variables 
(t = 70)

Collective variables 
(t = 75)

Collective variables 
(t = 80)

Collective variables serve as proxy for structural and conformational changes

In Situ Analysis of MD Trajectories

In situ analysis is beneficial since we can:

1. Capture outputs in memory at runtime as they are generated
2. Integrate simulation and analytics into complex workflows for runtime detection of changes in structural and temporal molecular properties
3. Annotate MD outputs and store the results of every step in the workflow
4. Design new data representations and extend unsupervised machine learning techniques to organise structural and temporal molecular properties
In Situ Analysis of MD Trajectories: A4MD

- Run n-Stride simulation steps
- MD code (e.g., GROMACS)
- Plumed
- Ingestor

MD Frame Generation

In-memory Staging Area (DataSpaces)

Data Storage

ML-inferred algorithms
Collective variables

Analytics representations and algorithms

In Situ Data Analytics

See https://analytics4md.org/ for more information regarding A4MD
In Situ Analysis of MD Trajectories: A4MD

MD Frame Generation

- Run n-Stride simulation steps
- MD code (e.g., GROMACS)
- Plumed
- Ingestor

Data Storage

- In-memory Staging Area (DataSpaces)
- Retriever

In Situ Data Analytics

- ML-inferred algorithms
  - Collective variables
  - Analytics representations and algorithms

This approach also allows to introduce feedback to the simulation

See https://analytics4md.org/ for more information regarding A4MD
An holistic approach that co-locates simulation and analysis can benefit from

- Natural integration with data streaming
- Massive task parallelism
- In-memory storage

However, we must find a way to integrate MD simulations with the analytics

**Step 1**
Leverage the *in situ* approach for data analytics

**Step 2**
Introduce support for the orchestration of multiple tasks
From One Trajectory to Many

MD Frame Generation
In Situ Data
Analytics

A4MD API
A4MD API

In-memory staging
DataSpaces

Inter-application data sharing
Parallel File System
From One Trajectory to Many

Predictions and Steering
• Understand and annotate dynamics in MD trajectories
• Enable on-the-fly tuning of MD workflows (i.e., stop, start, and fork MD jobs)
• Enhance adaptive sampling in MD trajectories

MD Frame Generation

In Situ Data Analytics

In-memory staging

Inter-application data sharing

DataSpaces

Parallel File System
From One Trajectory to Many

Predictions and Steering
- Understand and annotate dynamics in MD trajectories
- Enable on-the-fly tuning of MD workflows (i.e., stop, start, and fork MD jobs)
- Enhance adaptive sampling in MD trajectories

MD Frame Generation

In Situ Data Analytics

In-memory staging

DataSpaces

Inter-application data sharing

Parallel File System
From One Trajectory to Many

Predictions and Steering
- Understand and annotate dynamics in MD trajectories
- Enable on-the-fly tuning of MD workflows (i.e., stop, start, and fork MD jobs)
- Enhance adaptive sampling in MD trajectories

MD Frame Generation
- A4MD API

In Situ Data Analytics
- A4MD API

In-memory staging

Inter-application data sharing

DataSpaces

Parallel File System
Generalising the Workflow Engine: A4X

Multiple data sources and analysis modules can exchange information in situ through A4X.

Data Generation
- A4X API

Data Generation
- A4X API

In Situ Data Analytics
- A4X API

In Situ Data Analytics
- A4X API

In-memory staging

Inter-application data sharing

DataSpaces

Parallel File System
From HPC to BDA MD Simulations

An holistic approach that co-locates simulation and analysis can benefit from

- Natural integration with data streaming
- Massive task parallelism
- In-memory storage

However, we must find a way to integrate MD simulations with the analytics

**Step 1**
Leverage the *in situ* approach for data analytics

**Step 2**
Introduce support for the orchestration of multiple tasks
Convergence Lessons from Ensemble MD
Convergence Lessons from Ensemble MD

1. Simulations can act as “streaming devices”
   ○ They generate data that needs to be consumed
   ○ The generation needs to be orchestrated with the analysis

2. Annotation and data exchanges can benefit from the in-memory storage hierarchy

3. Management of an ensemble requires introducing elasticity (i.e. dynamic task allocation according to workload needs)

4. Programmability is key
   ○ MD workflows require interfaces between different programming languages and models

5. These are common challenges for all in situ analytics
   ○ We are generalising A4MD into A4x, and applying in situ approaches in other domains like neural network architecture search
Convergence Lessons from Ensemble MD

1. Simulations can act as “streaming devices”
   ○ They generate data that needs to be consumed
   ○ The generation needs to be orchestrated with the analysis

2. Annotation and data exchanges can benefit from the in-memory storage hierarchy
Convergence Lessons from Ensemble MD

1. Simulations can act as “streaming devices”
   - They generate data that needs to be consumed
   - The generation needs to be orchestrated with the analysis

2. Annotation and data exchanges can benefit from the in-memory storage hierarchy

3. Management of an ensemble requires introducing elasticity (i.e. dynamic task allocation according to workload needs)

4. Programmability is key
   - MD workflows require interfaces between different programming languages and models

5. These are common challenges for all in situ analytics
   - We are generalising A4MD into A4x, and applying in situ approaches in other domains like neural network architecture search
Convergence Lessons from Ensemble MD

1. Simulations can act as “streaming devices”
   ○ They generate data that needs to be consumed
   ○ The generation needs to be orchestrated with the analysis

2. Annotation and data exchanges can benefit from the in-memory storage hierarchy

3. Management of an ensemble requires introducing elasticity (i.e. dynamic task allocation according to workload needs)

4. Programmability is key
   ○ MD workflows require interfaces between different programming languages and models

5. These are common challenges for all in situ analytics
   ○ We are generalising A4MD into A4x, and applying in situ approaches in other domains like neural network architecture search
Convergence Lessons from Ensemble MD

1. Simulations can act as “streaming devices”
   ○ They generate data that needs to be consumed
   ○ The generation needs to be orchestrated with the analysis

2. Annotation and data exchanges can benefit from the in-memory storage hierarchy

3. Management of an ensemble requires introducing elasticity (i.e. dynamic task allocation according to workload needs)

4. Programmability is key
   ○ MD workflows require interfaces between different programming languages and models

5. These are common challenges for all in situ analytics
   ○ We are generalising A4MD into A4X, and applying in situ approaches in other domains like neural network architecture search
Scientific Case Studies: Geographically Distributed Water Forecasting
Hydrogeological Simulations for Water Forecasting

- Water resource management requires fast and reliable knowledge of very complex hydrogeological systems.
- Relies on real-time stochastic simulation of water profiles:
  - Multiscale non-linear processes and matrix operations from multi-physics models.
  - Input data from several geographically distributed sensors.
  - Severe deadlines (forecasts, status reports for emergencies).
Hydrogeological Simulations for Water Forecasting

HPC has increased the scale and complexity of the simulations:
- Multicore systems
- Distributed cluster computing
- Grid-like technologies

But HPC insufficient to cover the needs of a geographically distributed sensor network
From HPC to BDA Hydrogeological Simulations

Cloud-based deployments can benefit from

- Elasticity of computational power in case of emergency (hard deadlines)
- Cloud storage for archival purposes
- Natural integration with data streaming

However, we must find a way to make HPC simulations suitable for this scenario

**Step 1**
Transform the simulation into a data-centric application

**Step 2**
Integrate the data-centric simulation into a BDA platform suitable for cloud
Cloud-based deployments can benefit from

- Elasticity of computational power in case of emergency (hard deadlines)
- Cloud storage for archival purposes
- Natural integration with data streaming

However, we must find a way to make HPC simulations suitable for this scenario

**Step 1**
Transform the simulation into a data-centric application

**Step 2**
Integrate the data-centric simulation into a BDA platform suitable for cloud
Data-Centric Transformation of an HPC Application
Loosely-coupled and pleasingly parallelisable
Data-Centric Transformation of an HPC Application

Loosely-coupled and pleasingly parallelisable

Tightly-coupled and not pleasingly parallelisable
Data-Centric Transformation of an HPC Application

Loosely-coupled and pleasingly parallelisable

Tightly-coupled and not pleasingly parallelisable

BDA

HPC
Data-Centric Transformation of an HPC Application
Data-Centric Transformation of an HPC Application

We can use the Map-Reduce model to decouple each subsimulation!
Data-Centric Transformation of an HPC Application

Tightly-coupled and not pleasingly parallelisable

Loosely-coupled and pleasingly parallelisable

BDA

SELECT INDEPENDENT VARIABLE

SIMULATION PARAMETERS

SHARED FILES

DATABASE

INITIALISATION

DOMAIN DISTRIBUTION (PARALLELISE)

DATA ADAPTATION (MAP)

DATA ADAPTATION (MAP)

DATA ADAPTATION (MAP)

SIMULATION (MAP)

SIMULATION (MAP)

PARTIAL EVALUATION (REDUCE)

PARTIAL EVALUATION (REDUCE)

OUTPUT ANALYSIS (COLLECT)

OUTPUT FILES

HPC
Data-Centric Transformation of an HPC Application

We can apply a data-centric transformation methodology based on Map-Reduce if we can establish that:

1. There is at least one pleasingly parallelisable simulation domain, represented by an independent variable, $k$, used for indexing purposes
   - Examples: independent time-domain steps, a range of simulation parameters, etc.

2. Data can be arranged in two groups:
   - Data that is required by every subsimulation (e.g. common simulation parameters)
   - Data that is required for a specific value of $k$
Data-Centric Transformation of an HPC Application

We can apply a data-centric transformation methodology based on Map-Reduce if we can establish that:

1. There is at least one pleasingly parallelisable simulation domain, represented by an independent variable, $k$, used for indexing purposes
   - Examples: independent time-domain steps, a range of simulation parameters, etc.

2. Data can be arranged in two groups:
   - Data that is required by every subsimulation (e.g. common simulation parameters)
   - Data that is required for a specific value of $k$
Data-Centric Transformation of an HPC Application

We can apply a data-centric transformation methodology based on Map-Reduce if we can establish that:

1. There is at least one pleasingly parallelisable simulation domain, represented by an independent variable, $k$, used for indexing purposes
   - Examples: independent time-domain steps, a range of simulation parameters, etc.

2. Data can be arranged in two groups:
   - Data that is required by every subsimulation (e.g. common simulation parameters)
   - Data that is required for a specific value of $k$
From HPC to BDA Hydrogeological Simulations

Cloud-based deployments can benefit from

- Elasticity of computational power in case of emergency (hard deadlines)
- Cloud storage for archival purposes
- Natural integration with data streaming

However, we must find a way to make HPC simulations suitable for this scenario

**Step 1**
Transform the simulation into a data-centric application

**Step 2**
Integrate the data-centric simulation into a BDA platform suitable for cloud
A Generalist Architecture for HPC-BDA Applications

- Enable transparent access to existing BDA and HPC features
- Expose a unified data abstraction and operational model
- Build on existing runtimes
- Allow process-centric workloads to interact with BDA platforms and infrastructures
A Generalist Architecture for HPC-BDA Applications

- Enable transparent access to existing BDA and HPC features
- Expose a unified data abstraction and operational model
- Build on existing runtimes
- Allow process-centric workloads to interact with BDA platforms and infrastructures
A Generalist Architecture for HPC-BDA Applications

- Enable transparent access to existing BDA and HPC features
- Expose a unified data abstraction and operational model
- Build on existing runtimes
- Allow process-centric workloads to interact with BDA platforms and infrastructures
A Generalist Architecture for HPC-BDA Applications

- Enable transparent access to existing BDA and HPC features
- Expose a unified data abstraction and operational model
- **Build on existing runtimes**
- Allow process-centric workloads to interact with BDA platforms and infrastructures
A Generalist Architecture for HPC-BDA Applications

- Enable transparent access to existing BDA and HPC features
- Expose a unified data abstraction and operational model
- Build on existing runtimes
- Allow process-centric workloads to interact with BDA platforms and infrastructures
The Spark-DIY HPC-BDA Platform

We designed and implemented Spark-DIY as an example of this architecture

- **Runtimes:** Apache Spark and Do-It-Yourself block parallelism (DIY), built on MPI
- **Data abstraction:** mapping of resilient distributed dataset (RDD) and DIY block topology
- **Operational model:** Spark API with additional offload operator for DIY, plus additional features required for HPC (e.g. stateful operations)
- **Runtime Delegation:** Spark context for data distribution and I/O, Spark master for task generation

Example of Iterative Analysis

class KernelCallback extends OffloadDIYCallback {
    override def kernel() = { // Call kernel interface }
    override def input(ptr:Long) = { // Arrange data in memory }
    override def output(ptr:Long) = { // Retrieve data from memory }
}

def main() {
    // ...
    var dataset = sparkContext.parallelize(placeholder, numBlocks)
    for (i <- 0 to iterations)
        var offDDD = new OffloadDDD(dataset, numBlocks, sparkContext, stateful)
        var outRDD = offDDD.DIYoffload(new KernelCallback())

        // Run analysis on output RDD, which contains kernel results
    }
    // ...
}
Example of Iterative Analysis

class KernelCallback extends OffloadDIYCallback {
    override def kernel() = { // Call kernel interface }
    override def input(ptr:Long) = { // Arrange data in memory }
    override def output(ptr:Long) = { // Retrieve data from memory }
}

def main() {
    // ...
    var dataset = sparkContext.parallelize(placeholder, numBlocks)
    for (i <- 0 to iterations)
        var offDDD = new OffloadDDD(dataset, numBlocks, sparkContext)
        var outRDD = offDDD.DIYoffload(new KernelCallback())
        // Run analysis on output RDD, which contains kernel results
    // ...
}
Example of Iterative Analysis

```scala
class KernelCallback extends OffloadDIYCallback {
  override def kernel() = { // Call kernel interface }
  override def input(ptr: Long) = { // Arrange data in memory }
  override def output(ptr: Long) = { // Retrieve data from memory }
}

def main() {
  // ...
  var dataset = sparkContext.parallelize(placeholder, numBlocks)
  for (i <- 0 to iterations) {
    var offRDD = new OffloadRDD(dataset, numBlocks, sparkContext, stateful)
    var outRDD = offRDD.DIYoffload(new KernelCallback())
    // Run analysis on output RDD, which contains ...
  }
  // ...
}
```

Configure parallelisation for the entire workflow
Example of Iterative Analysis

```java
class KernelCallback extends OffloadDIYCallback{
  
  // Configure dataset, indicating if f will be immutable or not
  // (in this case, we want it to be stateful)

  def main() {
    // ...
    var dataset = sparkContext.parallelize(placeholder, numBlocks)
    for (i <- 0 to iterations)
      var offDDD = new OffloadDDD(dataset, numBlocks, sparkContext, stateful)
      var outRDD = offDDD.DIYoffload(new KernelCallback())

      // Run analysis on output RDD, which contains kernel results
    // ...
  }
}
Example of Iterative Analysis

class KernelCallback extends OffloadDIYCallback {
  override def kernel() = { // Call kernel interface }
  override def input(ptr:Long) = { // Arrange data in memory }
  override def output(ptr:Long) = { // Retrieve data from memory }
}

def main() {
  // ...
  var dataset = sparkContext.parallelize(placeholder, numBlocks)
  for (i <- 0 to iterations)
    var offDDD = new OffloadDDD(dataset, numBlocks, sparkContext, stateful)
    var outRDD = offDDD.DIYoffload(new KernelCallback()
      // Run analysis on output RDD, which contains kernel results
  }
  // ...
}
Cloud-based deployments can benefit from

- Elasticity of computational power in case of emergency (hard deadlines)
- Cloud storage for archival purposes
- Natural integration with data streaming

However, we must find a way to make HPC simulations suitable for this scenario

**Step 1**
Transform the simulation into a data-centric application

**Step 2**
Integrate the data-centric simulation into a BDA platform suitable for cloud
From HPC to BDA Hydrogeological Simulations

Cloud-based deployments can benefit from

- Elasticity of computational power in case of emergency (hard deadlines)
- Cloud storage for archival purposes
- Natural integration with data streaming

However, we must find a way to make HPC simulations suitable for this scenario

**Step 1**
Transform the simulation into a data-centric application

**Step 2**
Integrate the data-centric simulation into a BDA platform suitable for cloud
Convergence Lessons from Water Forecasting
Convergence Lessons from Water Forecasting

1. Geographically distributed applications can also appear in scientific domains
   ○ Sensor network devices create an architecture that is similar to the Fog model
Convergence Lessons from Water Forecasting

1. Geographically distributed applications can also appear in scientific domains
   ○ Sensor network devices create an architecture that is similar to the Fog model

2. Some workflows can be built with stages that map to either HPC and BDA paradigms
   ○ Need to share data efficiently
   ○ Platforms like Spark-DIY avoid the need to go to the PFS to share data between runtimes
Convergence Lessons from Water Forecasting

1. Geographically distributed applications can also appear in scientific domains
   ○ Sensor network devices create an architecture that is similar to the Fog model

2. Some workflows can be built with stages that map to either HPC and BDA paradigms
   ○ Need to share data efficiently
   ○ Platforms like Spark-DIY avoid the need to go to the PFS to share data between runtimes

3. Cloud infrastructures can assist in many ways
   ○ Resource elasticity (i.e. dynamic core allocation for emergency situations)
   ○ Distributed storage
Summary
Takeaways: The Convergence Problem

- Large-scale computing is in transition to a new generation of data-intensive applications
- There is a growing confluence between HPC and BDA
  - HPC applications produce Big Data
  - BDA is a growing consumer of HPC capabilities
  - More and more applications with mixed requirements
  - Infrastructures that tend to become hybrid

Upcoming applications will suffer the lack of an ideal environment able to handle their computing and data requirements
Takeaways: Open Challenges

- We try to aim for the best of both worlds: HPC performance with BDA flexibility
- There is no magic solution, for each use case we must
  - Prioritise features
  - Compromise metrics

HPC-BDA Convergence is now an established research area, but challenges remain when we consider the future of each ecosystem

- Data processing volumes and speed
- Upcoming exascale machines
- New research and industry needs
Convergence of High-Performance Computing and Big Data Analytics: Applications and Open Challenges

Silvina Caíno-Lores, PhD
scainolo@utk.edu