**Sources of Information**

In discussing the structure of Big Data Applications, let us first discuss the inevitably incomplete input that we used to do our analysis. We have gained of course quite a bit of experience from our research over many years but 3 explicit sources that we used were a recent use case survey by NIST from Fall 2013; a survey of data intensive research applications (Distributed Computing MetaPatterns) by Jha et al.; and study of members of data analytics libraries including R, Mahout and MLLib. We follow with a summary of first two sources.

The NIST Big Data Public Working Group (NBD-PWG) was launched in June 2013 with a set of working groups covering Big Data Definitions, Taxonomies, Requirements, Security and Privacy Requirements, Reference Architectures White Paper Survey, Reference Architectures, Security and Privacy Reference Architectures and Big Data Technology Roadmap. The Requirements working group gathered 51 use cases from a public call and then analyzed in terms of requirements of a reference architecture. Here we will look at them differently to identify common patterns and characteristics which can be used to guide and evaluate Big Data hardware and software.

The use cases are organized into nine broad areas as follows, with the number of associated use cases in parentheses:

* Government Operation (4)
* Commercial (8)
* Defense (3)
* Healthcare and Life Sciences (10)
* Deep Learning and Social Media (6)
* The Ecosystem for Research (4)
* Astronomy and Physics (5)
* Earth, Environmental and Polar Science (10)
* Energy (1)

Note that the majority of use cases come from research applications but commercial, defense and government operations have some coverage. A template was prepared by the requirements working group, which allowed experts to categorize each use case by 26 features that included

Use case Actors/Stakeholders and their roles and responsibilities; use case goals and description. Specification of current analysis covering compute system, storage, networking and software. Characteristics of use case Big Data with Data Source (distributed/centralized), Volume (size), Velocity (e.g. real time), Variety (multiple datasets, mashup), Variability (rate of change). The so-called Big Data Science (collection, curation, analysis) with Veracity (Robustness Issues, semantics), Visualization, Data Quality (syntax), Data Types and Data Analytics. These detailed specifications were complemented by broad comments including Big Data Specific Challenges (Gaps), Mobility issues, Security & Privacy Requirements and identification of issues for generalizing this use case.

The complete set of 51 responses with in addition a summary from the working group of applications, current status and futures as well as extracted requirements can be found in []

Need a discussion of Jha et al.

**Lessons from Parallel Computing**

Before discussing features and patterns of Big Data applications, it is instructive to consider the better understood parallel computing situation. Here the application requirements have been captured in many ways

1. **Benchmark Sets.** These vary from full applications [Kuck] to kernels or mini-applications such as the NAS Parallel Benchmarks[] or Parkbench[http://www.netlib.org/parkbench/] with the Top500 pacing application Linpack (HPL) particularly well known. The new sparse HPCG conjugate gradient benchmark is notable. Note benchmarks can be specified via explicit code and/or specified by a “pencil and paper specification” that can be optimized in any way for a particular platform.
2. **Patterns or Templates.** These can be similar to benchmarks but with different goals such as providing a generic framework that can be modified by users with details of their application as in Template book [Siam, ?recent UIUC]. Alternatively they can be aimed at illustrating different applications as in original Berkeley Dwarves [].

In this paper, our approach is nearest that of the Dwarves and one motivation for us calling our work the Big Data Ogres. In looking at this previous work, we note that benchmarks often cover a variety of different application aspects and are accompanied by principles or folk lore that can guide the writing of parallel code or designing suitable hardware and software. For example, Data locality and cost of data movement, sparseness, Amdahl’s law, communication latency and bisection bandwidth and scaled speedup are associated with substantial folklore.

The famous NAS Parallel Benchmarks NPB consists of MG: Multigrid, CG: Conjugate Gradient, FT: Fast Fourier Transform, IS: Integer sort, EP: Embarrassingly Parallel, BT: Block Tridiagonal, SP: Scalar Pentadiagonal, and LU: Lower-Upper symmetric Gauss Seidel, are pretty uniform. With the exception of EP which is an application class, the other members are typical constituents of a low level library for parallel simulations. On the other hand the Berkeley dwarves are Dense Linear Algebra , Sparse Linear Algebra, Spectral Methods, N-Body Methods, Structured Grids, Unstructured Grids, MapReduce, Combinational Logic, Graph Traversal, Dynamic Programming, Backtrack and Branch-and-Bound, Graphical Models and Finite State Machines. The dwarves are not exact kernels but describe problem from different points of view including programming model (MapReduce), numerical method (Grids, Spectral method), kernel structure (dense or sparse linear algebra), algorithm (dynamic programming) and application class (N-body) etc. We think that it is inevitable that both parallel computing and Big Data cannot be characterized with a single criteria and so we introduce multiple facets for our Ore characterization.

Add Jha et al. lessons

**Properties of the 51 use cases**

Tables 1 to 3 summarize characteristics of the 51 use cases which we will combine with other input for the Ogres. Note that Big Data and parallel programming are intrinsically linked as any Big Data analysis is inevitably processed in parallel. Parallel computing is almost always implemented by dividing the data between processors (data decomposition); the richness here is illustrated in Table 1 which lists the members of space that is decomposed for different use cases. In Table 2, we identify 15 use case features that will be used later as components of the Ogre facets. The second column of Table 2 lists our estimate of the number of use cases that illustrate this feature; note these are not exclusive so any one use case will illustrate many features.

|  |
| --- |
| Table 1: What is Parallelism Over for NIST Use Cases? |
| General Class | **Examples** |
| People | Users (but see below) or Subjects of application and often both |
| Decision makers | Researchers or doctors (users of application) |
| Items | Experimental observations |
| Contents of online store |
| Images or “Electronic Information nuggets” |
| EMR: Electronic Medical Records (often similar to people parallelism) |
| Protein or Gene Sequences |
| Material properties, Manufactured Object specifications, etc., in custom dataset |
| Modelled entities | Vehicles and people |
| Sensors | Internet of Things |
| Events | Detected anomalies in telescope, credit card or atmospheric data |
| Graph Nodes | RDF databases |
| Regular Nodes | Simple nodes as in a learning network |
| Information Units | Tweets, Blogs, Documents, Web Pages, etc. and characters/words in them |
| Files or data | To be backed up, moved or assigned metadata |
| Particles/cells/ mesh points | Used in parallel simulations |

|  |
| --- |
| Table 2: Some Features of NIST Use Cases |
| Abbreviation | **#** | **Description** |
| PP | 26 | Pleasingly Parallel or Map Only |
| MR | 18 | Classic MapReduce MR (add MRStat below for full count) |
| MRStat | 7 | Simple version of MR where key computations are simple reduction as found in statistical averages such as histograms and averages |
| MRIter | 23 | Iterative MapReduce or MPI |
| Graph | 9 | Complex graph data structure needed in analysis  |
| Fusion | 11 | Integrate diverse data to aid discovery/decision making; could involve sophisticated algorithms or could just be a portal |
| Streaming | 41 |  Some data comes in incrementally and is processed this way |
| Classify | 30 | **Classification:** divide data into categories |
| S/Q | 12 | **Index**, Search and Query |
| CF | 4 | **Collaborative Filtering** for recommender engines |
| LML | 36 | **Local Machine Learning** (Independent for each parallel entity) |
| GML | 23 | **Global Machine Learning:** Deep Learning, Clustering, LDA, PLSI, MDS, Large Scale Optimizations as in Variational Bayes, Lifted Belief Propagation, Stochastic Gradient Descent, L-BFGS, Levenberg-Marquardt . Sometimes call EGO or Exascale Global Optimization with scalable parallel algorithm |
|  | 51 | **Workflow**: Universal so no label |
| GIS | 16 | **Geotagged data** and often displayed in ESRI, Microsoft Virtual Earth, Google Earth, GeoServer etc. |
| HPC | 5 | Classic **large-scale simulation** of cosmos, materials, etc. generating (visualization) data |
| Agent | 2 | Simulations of models of data-defined macroscopic entities represented as **agents** |

It’s important to note that machine learning is commonly used but there is an interesting distinction between what are termed Local (LML) and Global machine learning (GML) in Table 2. In LML, there is parallelism over items of Table 1 and machine learning is applied separately to each item; needed machine learning parallelism is limited and is typified by use of accelerators (GPU). In GML, the machine learning is applied over the full dataset with MapReduce, MPI or equivalent. Typically GML comes from maximum likelihood or χ2 with a sum over the data items – documents, sequences, items to be sold, images etc. and often links (point-pairs). Usually GML is a sum of positive numbers as in least squares and is illustrated by algorithms like PageRank, clustering/community detection, mixture models, topic determination, Multidimensional scaling, and (Deep) Learning Networks. Somewhat quixotically, GML can be termed Exascale Global Optimization or EGO. The difference between LML and GML is illustrated in Table 3, which contrasts 9 of the 51 NIST use cases that involve image based data. For example use case 18 with light source data is largely independent machine learning on each image from the source i.e. LML. In contrast deep learning in use case 26, is constructing a learning network integrating all the images.

|  |
| --- |
| Table 3: 9 Image-based NIST Use Cases |
| Use Case | **Title** | **Application** | **Features** |
| 17 | Pathology Imaging/ Digital Pathology | Moving to terabyte size 3D images, Global Classification | PP, LML, MR for search |
| 18 | Light sources | Biology and Materials | PP, LML |
| 26  | Large-scale Deep Learning | Stanford ran 10 million images and 11 billion parameters on a 64 GPU HPC; vision (drive car), speech, and Natural Language Processing | GML |
| 27 | Organizing large-scale, unstructured collections of photos | Fit position and camera direction to assemble 3D photo ensemble | GML |
| 36 | Catalina Real-Time Transient Synoptic Sky Survey (CRTS) | Processing of individual images for events based on classification of image structure (GML) | PP, LML |
| 43 | Radar Data Analysis for CReSIS Remote Sensing of Ice Sheets | Identify glacier beds and snow layersSee GML when one addresses full ice sheet | PP, LML moving to GML  |
| 44  | UAVSAR Data Processing,  | Find and display slippage from radar images. Includes Data Product Delivery, and Data Services | PP |
| 45, 46 | Analysis of Simulation visualizations | Find paths, classify orbits, classify patterns that signal earthquakes, instabilities, climate, turbulence | PP LML ?GML |

**Properties of other use cases**

Other NIST (GCF to do)

and Jha et al.

**The four Facets of the Big Data Ogres**

We introduce 4 facets or classification dimensions to categorize Big data applications. These are Problem architecture and style, Computational features, Data Source or Style and Analytics Algorithm/Kernel. There are of course other ways of looking at the Ogres and our work should be treated as an initial suggestion for further discussion.

**Problem Architecture and Style Facet of Ogres**

|  |  |
| --- | --- |
|  | Table 4: Problem Architecture Facet of Ogres (Meta or Macro Pattern) |
| Pleasingly Parallel | as in BLAST, Protein docking, some (bio-)imagery including Local Analytics or Local Machine Learning with pleasingly parallel filtering, as in light source data, radar images  |
| Classic MapReduce | Search, Index and Query and Classification algorithms like collaborative filtering |
| GML | Global Analytics or Global Machine Learning requiring iterative programming models |
| Graph | Problem set up as a graph as opposed to vector, grid |
| SPMD | SPMD (Single Program Multiple Data) |
| BSP | Bulk Synchronous Processing: well-defined compute-communication phases |
| Fusion or Workflow | Knowledge discovery often involves fusion of multiple methods. All applications often involve orchestration (workflow) of multiple components |
| Agents | As used in epidemiology, discrete event simulations etc. Swarm approaches |

Expectation Maximization and Steepest descent methods often seen in LML or GML

**Computational features Facet of Ogres**

|  |
| --- |
| Table 5: Computational Features Facet of Ogres |
| Flops per byte: important for performance |
| Communication Interconnect requirements;  |
| Is application (graph) constant or dynamic? |
| Most applications consist of a set of interconnected entities; is this regular as a set of pixels or is it a complicated irregular graph? |
| Is communication BSP or Asynchronous? In latter case shared memory may be attractive; |
| Are algorithms Iterative or not? |
| Data Abstraction: key-value, pixel, graph, vector, HDF5 etc. |
| Are data points in metric or non-metric spaces?  |
| Core libraries needed: matrix-matrix/vector algebra, conjugate gradient, reduction, broadcast ….  |

Composition multiple styles

**Data Source or Style Facet of Ogres**

|  |
| --- |
| Table 6: Data Source and Style Facet of Ogres |
| SQL |
| NOSQL based |
| Other Enterprise data systems (10 examples from NIST [])  |
| Set of Files (as managed in iRODS) |
| Internet of Things |
| Streaming  |
| HPC simulations |
| Involve GIS (Geographical Information Systems) |
| Before data gets to compute system, there is often an initial data gathering phase which is characterized by a block size and timing. Block size varies from month (Remote Sensing, Seismic) to day (genomic) to seconds or lower (Real time control, streaming) |
| There are storage/compute system styles: Shared, Dedicated, Permanent, Transient |
| Other characteristics are needed for permanent auxiliary/comparison datasets and these could be interdisciplinary, implying nontrivial data movement/replication |

**Analytics Algorithm/Kernel Facet of Ogres**

|  |
| --- |
| Table 7: Core Analytics Facet of Ogres (microPattern) |
| Pleasingly Parallel (Map Only) or Local Machine Learning: ~any algorithm |
| Map-Reduce |
| Search, Query, Index: Dominant commercial use and important in Science with less users |
| Recommender Systems including Collaborative filtering: Dominant commercial use, Little Science |
| Summarizing statistics (MRStat) as in LHC Data analysis (histograms) |
| Linear Classifiers: Bayes, Random Forests |
| Global Analytics – Nonlinear Solvers (Structure depends on Objective Function) |
| Stochastic Gradient Descent SGD |
| (L-)BFGS approximation to Newton’s Method |
| Levenberg-Marquardt solver |
| Global Analytics – Map-Collective (See Mahout, MLlib) |
| Outlier Detection |
| Clustering (many methods) |
| Mixture Models, LDA (Latent Dirichlet Allocation), PLSI (Probabilistic Latent Semantic Indexing) |
| SVM and Logistic Regression |
| PageRank (find leading eigenvector of sparse matrix) |
| SVD (Singular Value Decomposition) |
| MDS (Multidimensional Scaling) |
| Learning Neural Networks (Deep Learning) |
| Hidden Markov Models |
| Global Analytics – Map-Communication (targets for Giraph) |
| Graph Structure (Communities, subgraphs/motifs, diameter, maximal cliques, connected components) |
| Network Dynamics - Graph simulation Algorithms (epidemiology) |
| Global Analytics – Asynchronous Shared Memory |
| Graph Structure (Betweenness centrality, shortest path) |

**Hardware and Software Architecture Issues**

|  |
| --- |
| Table 8: Distinctive Software/Hardware Architectures for Data Analytics |
| 1 | Pleasingly Parallel (Map Only) | Includes local machine learning (LML) as in parallel decomposition over items and apply data processing to each item. Hadoop could be used but also other High Throughput Computing or Many task tools |
| 2 | Classic MapReduce | Includes search applications and those using collaborative filtering and motif finding implemented using classic MapReduce (Hadoop) |
| 3 | Iterative Map-Collective | Iterative MapReduce using Collective Communication as needed in clustering – Hadoop with Harp, Spark etc. |
| 4 | Iterative Map-Communication | Iterative MapReduce such as Giraph with point-to-point communication and includes most graph algorithms such as maximum clique, connected component, finding diameter, community detection). Vary in difficulty of finding partitioning (classic parallel load balancing) |
| 5 | Shared Memory | Thread-based (event driven) graph algorithms such as shortest path and Betweenness centrality |

In table 8, we present 5 distinct problem architecture that map into 5 distinct system architectures which seem to cover the Ogres and their facets. 8.5 is the shared memory architecture needed for some graph algorithms that perform better here and also for some large memory applications. The central architectures are 8.1 to 8.4 which correspond exactly to the four forms of MapReduce that we have presented elsewhere [] but are summarized in figure 1. Note this only describes some core features of the facets in tables 4 and 5. There are many other issues that need to be addressed including support of workflow and the data systems captured in the facets of table 6.

*Figure 1: The Four forms of MapReduce that correspond to the four architectures of Table 8.1-8.4*

**Comparison between Data Intensive and Simulation Problems**

We can use the Ogre analysis and the data analytics architectures to compare data intensive and simulation applications. There are some clear similarities with from table 4, “Pleasingly parallel” (8.1), BSP and SPMD common in both arenas. However the Classic MapReduce architecture (8.2) is a major big data paradigm but much less common in simulations with one example between the execution of multiple simulations (as in Quantum Monte Carlo) followed by a reduce operation to collect the results of different simulations. The Iterative Map-Collective architecture (8.3) is common in much Big Data analytics as in clustering where there is no local graph structure and the parallel algorithms involve large scale collectives but no point to point communication. The same structure is seen in N-body (long range force) or other “all-pairs” simulations without the locality typical from discretizing differential operators.

Many simulation problems have the Map-Communication (8.4) architecture with many smallish point-to-point messages coming from local interactions between points defining system to be simulated. The importance of sparse data structures and algorithms is well understood in simulations and is seen in some Big Data problems such as PageRank, which calculates the leading eigenvector of the sparse matrix formed by internet site links. Other Big Data sparse data structures are seen in user-item ratings and bags of words problem. Most items are rated by few users and many documents contain a small fraction of the word vocabulary. However important data analytics involve full matrix algorithms and for example recent papers [] on a new Multi-Dimensional Scaling method use conjugate gradient solvers with full matrices as opposed to the new sparse conjugate gradient benchmark HPCG being developed for supercomputer (Top500) evaluations.

Note that there are similarities between some Big Data graph problems and particle simulations with a strange cutoff force. Both use the Map-Communication architecture and the links in a Big Data graph are equivalent to strength of force between the graph nodes considered as particles. In this analogy, many Big Data problems are “long range force” corresponding to a graph where all nodes are linked to each other. As in simulation case, these O(N2) problems are typically very compute intense but straightforward to parallelize efficiently. It is interesting to consider the analogue of the “fast multipole” methods for the fully connected Big Data problems which can dramatically improve the performance to O(N) or O(NlogN) []. Finally note the network connections used in deep learning are sparse but in recent image interpretation studies [Coates], the network weights are block sparse (corresponding to links to pixel blocks) and can be formulated as full matrix operations with GPUs and MPI running efficiently with these blocks.

The final architecture 8.5 (Shared Memory) is important in some applications but not heavily used in either simulations or Big Data.

**Implementing Big Data**

|  |
| --- |
| Table 9: Kaleidoscope of (Apache) Big Data Stack (ABDS) and HPC Technologies |
|

|  |
| --- |
| Cross-Cutting Functionalities |
| Message Protocols: Thrift, Protobuf |
| Distributed Coordination: Zookeeper, JGroups |
| Security & Privacy |
| Monitoring: Ambari, Ganglia, Nagios, Inca |

 |

|  |
| --- |
| **Workflow-Orchestration:** Oozie, ODE, Airavata, OODT (Tools), Pegasus, Kepler, Swift, Taverna, Trident, ActiveBPEL, BioKepler, Galaxy, IPython  |
| **Application and Analytics:** Mahout , MLlib , MLbase, CompLearn, R, Bioconductor, ImageJ, Scalapack, PetSc |
| **High level Programming:** Hive, HCatalog, Pig, Shark, MRQL, Impala, Sawzall |
| **Basic Programming model and runtime**, **SPMD, Streaming, MapReduce, MPI:** Hadoop, Spark, Twister, Stratosphere, Tez, Hama, Storm, S4, Samza, Giraph, Pregel, Pegasus |
| **Inter process communication Collectives, point-to-point, publish-subscribe:** Hadoop, Spark, Harp, MPI, Netty, ZeroMQ, ActiveMQ, QPid, Kafka, Kestrel |
| **In-memory databases/caches:** GORA (general object from NoSQL), Memcached, Redis (key value), Hazelcast, Ehcache |
| **Object-relational mapping:** Hibernate, OpenJPA and JDBC Standard |
| **Extraction Tools:** UIMA, Tika |
| **SQL:** Oracle, MySQL, Phoenix, SciDB |
| **NoSQL:** HBase, Accumulo, Cassandra, Solandra, MongoDB, CouchDB, Lucene, Solr, Berkeley DB, Azure Table, Dynamo, Riak, Voldemort. Neo4J, Yarcdata, Jena, Sesame, AllegroGraph, RYA |
| **File management:** iRODS |
| **Data Transport:** BitTorrent, HTTP, FTP, SSH, Globus Online (GridFTP) |
| **Cluster Resource Management**: Mesos, Yarn, Helix, Llama, Condor, SGE, OpenPBS, Moab, Slurm, Torque |
| **File systems:** Swift, Cinder, Ceph, FUSE, Gluster, Lustre, GPFS, GFFS |
| **Interoperability:** Whirr, JClouds, OCCI, CDMI |
| **DevOps:** Puppet, Chef, Ansible, Boto, Libcloud, Cobbler, CloudMesh |
| **IaaS Management from HPC to hypervisors:** OpenStack, OpenNebula, Eucalyptus, CloudStack, vCloud, Amazon, Azure, Google |

 |

We have described elsewhere how we propose to implement Big Data applications exploiting the HPBDS architecture sketched in Table 9 []. This combines the best practice commercial Big Data software with an emphasis on Apache projects with HPC subsystems. Table 9 illustrates by green shading those layers where HPC adds significant value to the Apache stack ABDS. Note that high performance communication is known to be critical for simulations but it is also essential for many science big data applications. Commercial applications have large “search” (8.2) components corresponding to the huge number of users accessing commercial Big Data systems. In science, this step is necessary – especially for good data management – but is a much lower fraction of system use as the number of scientists accessing data is much lower than number of users of commercial Big Data.