|  |  |
|---|---|
| **Date:**<br>**January 12, 2013** | **SOLUTION ARCHITECTURE- ARBITRARILY CONNECTED GRAPHS**<br>**FOR EXTREME-SCALE COMPUTING RESEARCH AND DEVELOPMENT (FAST FORWARD) STORAGE AND I/O** |

|  |  |
|---|---|
| LLNS Subcontract No. | B599860 |
| Subcontractor Name | Intel Federal LLC |
| Subcontractor Address | 2200 Mission College Blvd.<br>Santa Clara, CA 95052 |

**NOTICE:** THIS MANUSCRIPT HAS BEEN AUTHORED BY INTEL UNDER ITS SUBCONTRACT WITH LAWRENCE LIVERMORE NATIONAL SECURITY, LLC WHO IS THE OPERATOR AND MANAGER OF LAWRENCE LIVERMORE NATIONAL LABORATORY UNDER CONTRACT NO. DE-AC52-07NA27344 WITH THE U.S. DEPARTMENT OF ENERGY. THE UNITED STATES GOVERNMENT RETAINS AND THE PUBLISHER, BY ACCEPTING THE ARTICLE OF PUBLICATION, ACKNOWLEDGES THAT THE UNITED STATES GOVERNMENT RETAINS A NON-EXCLUSIVE, PAID-UP, IRREVOCABLE, WORLD-WIDE LICENSE TO PUBLISH OR REPRODUCE THE PUBLISHED FORM OF THIS MANUSCRIPT, OR ALLOW OTHERS TO DO SO, FOR UNITED STATES GOVERNMENT PURPOSES. THE VIEWS AND OPINIONS OF AUTHORS EXPRESSED HEREIN DO NOT NECESSARILY REFLECT THOSE OF THE UNITED STATES GOVERNMENT OR LAWRENCE LIVERMORE NATIONAL SECURITY, LLC.

**© 2014 Intel Corporation**

## Table of Contents

## Revision History

| Date | Revision | Author |
|---|---|---|
| 2012-01-12 | 1.0 Draft for Review | Arnab Paul, Jaewook Yu, Ted Willke, Intel Corporation |
|  |  |  |
|  |  |  |
|  |  |  |

## Definitions

The following is a list of definitions and terms that we use throughout the document.

- **Arbitrarily Connected Graph (ACG)**: A graph with arbitrary edge relationships. The graph may be a tree, bipartite, undirected, directed, or any number of types. In any case, it will not be complete. Many graphs that model natural structures and real-world phenomena are arbitrarily structured. Many of them are scale-free, and some exhibit small-world and clustering characteristics.

- **ACG Ingress**: The process of constructing and loading an ACG into the exascale system. The ACG ingress process comprises the Big Data-ACG bridge in this research. The graph will be constructed by applying extract and transform rules to large unstructured and semi-structured datasets.

- **Computational Kernel**: The application framework that supports the exascale structured machine learning and graph analytics. In this research, the computational kernel is based on GraphLab, an asynchronous distributed graph-parallel computational framework. GraphLab provides an in-memory data structure model, computational scheduling and synchronization, and a data consistency model.

- **Big Data Analytics (BDA)**: Big data analytics is the process of discovering latent patterns, understanding unknown correlations, or extracting meaningful information from data sets of which size and complexity are beyond the ability of traditional database management or data processing applications to process [1]. Some examples of big data include traffic sensory data (e.g., climate, traffic, etc.), stock and commercial transactions, social interaction data, and digitalized media (e.g., pictures and videos).

- **Big Data Graph**: An ACG with associated "network information" derived from a Big Data corpus. The network information is largely comprised of arbitrarily-typed vertex and edge data.

- **Gather, Apply, Scatter (GAS) model of computation**: The vertex-programming model defined by GraphLab (see *http://graphlab.org/home/abstraction*). A large number of vertex programs can be decomposed into Gather, Apply, and Scatter phases. In the Gather phase, the vertex program collects intermediate updates from neighboring vertex programs. During the Apply phase, it merges them with its own and updates variables. Finally, updates are asynchronously sent out to neighboring vertices during the Scatter phase.

- **Giant Component**: A giant component in a graph is a connected component that contains a constant fraction of the entire graph's vertices (http://en.wikipedia.org/wiki/Giant_component). Random and scale-free graphs tend to have giant components and their existence is important in proving many properties (such as stochastic convergence in percolation theory) about large graphs that are crucial to the success of many algorithms.

- **Large Scale Machine Learning (LSML)**: Machine Learning (ML), a branch of artificial intelligence, is about the construction and study of systems that can learn from data [2]. LSML refers to the scaling of machine learning algorithms to a compute cluster using parallel or distributed approaches.

- **Network Information**: Arbitrarily-typed data structures associated with vertices and edges.

- **Sub-Partition**: A graph-partition is typically represented as an ordered list (adjacency lists, edge lists etc.). A sub-partition is the further division of a partition into smaller portion. When a partition is too large for memory or thread processing, it may be divided into sub-partitions.
- **Real-World Graphs**: Graphs that represent relationships that arise from real world datasets.

- **Synthetic Graphs**: Graphs that are artificially generated by human or computer.

- **Vertex Program**: The execution thread that performs the vertex-based algorithm and GAS operation. Vertex programs are executed in parallel on each vertex and can interact with neighboring vertices (http://graphlab.org/home/abstraction/).

# 1 Introduction

## 1.1 ACG Applications for Exascale

The goal of this work is to investigate ACG-based Large-Scale Machine Learning (LSML) and graph analytics applications for exascale. In particular, we are interested in how such applications perform on the exascale storage and I/O subsystem under development and how they can best utilize the new architecture. ACG structures arise in many important contexts, including social network graphs [3, 4], topological maps of the internet [5], hyper-linked structures of the web [6], email networks [7], and the organization of human language [8]. These application contexts are assuming increasing importance in several fields, ranging from commercial space [9] to health and life-sciences [10] to national security [11].

ACG-based problems generate computational and I/O workloads that differ from those generated by the traditional Bulk Synchronous Processing (BSP)-type algorithms that are commonly used in high-performance computing. Although such ACG applications have been widely deployed and studied on commercial off-the-shelf systems, their deployment on supercomputers remains limited. In this proposal, we will investigate the suitability of the proposed storage and I/O subsystem for several applications involving ACG data models. Our observations will provide valuable feedback to the exascale storage and I/O architects, as well as to researchers of algorithms and computational models in both Big Data and HPC.

## 1.2 Arbitrarily Connected Graph-Related Challenges in Big Data and Large-Scale Machine Learning

Systems that perform well on algorithms that operate on random and complete graph structures often show performance degradation when applied to the ACG-like structures that appear in many real-world problems because these graphs follow power-law degree distributions.  Power-law graphs do not lend themselves to straightforward partitioning, since there are no clean cuts [9, 10], and poor partitioning quality leads to increased network communications and imbalances in compute and memory utilization. And, even if the partitioning quality is high and resource utilization balanced, the rate of iteration for each vertex program varies significantly and suffers from significant slowdown when implemented with BSP since most programs spend a large amount of time waiting for stragglers at synchronization boundaries. We believe these challenges can be overcome by extending the current algorithms for power-law graph partitioning in GraphBuilder to the ACG ingress process and by adapting GraphLab's asynchronous parallel

**The information on this page is subject to the use and disclosure restrictions provided on the cover page to this document. Copyright 2014, Intel Corporation.**

execution model [11, 12] to exascale and extending it to take advantage of data synchronization and consistency features of the storage and I/O subsystem.

We also believe that current HPC graph benchmarks, such as Graph500 [13], do not adequately capture the challenges associated with Big Data Graph ingress processing and Large-Scale Machine Learning. Although these benchmarks can generate ACGs, they do not specify power-law graphs and ignore the challenges associated with attached network information. Additionally, the benchmarks do not include any popular machine learning algorithms. But we believe that our development of a synthetic Big Data graph generator and several applications that incorporate the latest LSML algorithms will serve as a means to realistically benchmark the Big Data-HPC bridge ingress process and run-time storage and I/O performance.

### 1.3 Interfacing to existing libraries with the exascale stack

To maximize productivity, we plan to use the latest version of GraphLab, a computational kernel that provides many of the services we need (http://graphlab.org), such as asynchronous execution, graph data structure representation, data consistency, function updates, and synchronization. GraphLab requires its input to be in the form of a graph with associated network information. We will construct the graph in the ACG ingress process using GraphBuilder [14], a scalable graph construction library for Apache Hadoop capable of performing extract-transform-load-style preprocessing of semi-structured data. GraphBuilder will also partition the graph for exascale parallelization. To enable GraphLab and GraphBuilder to run on top of the exascale storage and I/O stack, we will prototype a new scalable HDF5 adaptation layer (HAL). Using HDF5 for data representation, our computational kernel will enable the I/O dispatcher to efficiently distribute ACGs and their partitions across storage servers. We anticipate such partitioning will result in efficient I/O streaming and pre-staging into NVRAM on IO nodes (IONs).

**The information on this page is subject to the use and disclosure restrictions provided on the cover page to this document.**
**Copyright 2014, Intel Corporation.**

## 2 Solution Requirements

We plan to modify available open source software for the computational kernel and the ACG-ingress process. The software must be modified to take maximum advantage of the proposed exascale storage and I/O subsystem. We must also develop several test applications that will exercise the Big Data-HPC bridge and storage and I/O subsystem. Finally, we must develop a synthetic Big Data graph generator that will allow us to study the capabilities of the system under a range of realistic loads.

### 2.1 HDF5 Adaptation Layer (HAL) for Graph Representation

#### 2.1.1 [Mandatory] HAL must translate commonly-used graph representations into the HDF5 data model

The computational framework and ingress process that we plan to use do not currently support HDF5. To address this, we must build an HDF5 adaptation layer (HAL) for translating graph structures to and from HDF5 representation into formats useable by the computational kernel. The HAL will use proposed extensions to HDF5 such as support for transactions, and will inform extensions in other areas so that ACGs can be represented effectively as HDF5 objects with usage hints that allow for high-performance storage and retrieval of the ACG vertex, edge, and network information.

#### 2.1.2 [Desirable] HAL must support conversion between different graph representations

Depending on application needs, the HAL may provide a means of changing the graph’s storage representation from edge list to adjacency list and from adjacency list to adjacency matrix, and so on.

#### 2.1.3 [Mandatory] HAL must associate additional network information with vertices and edges

In addition to information pertaining to topology or connectivity, our applications require that network information be attached to the graph. We must represent such data in HDF5, and maintain associations with the underlying graphs using the current or expanded HDF5 data types that will be provided in the HDF5 APIs.

### 2.2 [Mandatory] Big Data-HPC Bridge for ACG Ingress

There are two primary requirements for this component. First, it must have the ability to parse and extract graph structures and key network information from unstructured or semi-structured datasets. Second, it must represent, store, and partition these graphs into efficient HDF5 structures that can be used by LSML and BDA algorithms.

**The information on this page is subject to the use and disclosure restrictions provided on the cover page to this document. Copyright 2014, Intel Corporation.**

## 2.3 Computational Kernel for ACG Applications

This layer must provide systems-level abstraction (i.e., application framework) for LSML and BDA algorithms. It must include an in-memory data schema, and primitives for asynchronous updates, distributed shared objects manipulation, computational scheduling, and data synchronization. Additionally, the kernel must interface with the exascale storage and I/O subsystem.

### 2.3.1 [Mandatory] Computational kernel must support asynchronous vertex updates

ACG-based applications need frequent updates between compute nodes; these must be done asynchronously, else faster vertex programs will often block on the slower ones (the power-law degree distribution of the ACGs can create such imbalance).

### 2.3.2 [Desirable] Computational kernel may use the data sharing features exposed through HDF5 for efficient data sharing

Our computational kernel already provides data consistency and signaling primitives, but we may explore additional data-sharing features provided by the I/O stack. For example, a commonly encountered scenario is that a set of processes, not necessarily running on different nodes, are trying to access a shared data-object, such as a table of global variables in a somewhat restricted manner – the parallel updates do not overlap, i.e., done on independent variables (but part of a single larger object) while the processes reading data may want to read the entire object in a consistent manner. Typically this can happen in memory, but at exascale, the updates may be forced to take place out-of-core, and HAL should efficiently use HDF layer to support such concurrency.

### 2.3.3 [Desirable] The computational kernel may support the run-time loading of portions of graph partitions that do not fit into compute node memory

Some graph structures and network information may not fit entirely into compute node memory. In these cases, the graph partitions may be partially loaded as a sub-partition. The computational kernel would need to schedule the cascade of load-compute-unload operations and the resultant I/O. Whenever possible, the kernel would make use of efficient indexing of sub-partitions, bundle multiple output I/O streams, and optimize the sequence of transactions.

### 2.3.4 [Desirable] The computational kernel may generate hints for data usages that enable efficient prefetching and caching

Many graph-based computations will need to access graph objects randomly, while some will exhibit temporal and/or spatial locality. This is, in part, a function of the computational algorithm. These patterns will become recognizable over the course

**The information on this page is subject to the use and disclosure restrictions provided on the cover page to this document.**
**Copyright 2014, Intel Corporation.**

of our studies, and may be used to generate appropriate prefetching and caching hints for the HDF5 and IOD layers, resulting in higher effective I/O throughput.

#### 2.3.5 [Desirable] The computational kernel may utilize the HDF5 analysis shipper

Rather than bringing relatively large ACG and/or network information from storage to a compute node, the kernel may use the HDF5 extension APIs to ship out the appropriate script to be executed on I/O nodes or storage servers and wait for the result.

#### 2.3.6 [Desirable] Additional investigations

We may pursue the following additional investigations, if time permits:

- New algorithms for exascale power-law graph partitioning
- New compute scheduling engines that are better able to take advantage of the storage and I/O subsystem
- Use transactional I/O to prevent write-write races between adjacent update functions which are modifying the data within a vertex and scheduling future execution on other vertices.
- Bundling multiple updates into a single transaction for performance improvement
- New graph storage data representations

### 2.4 [Mandatory] Application testing

We must investigate the suitability of the exascale storage and I/O stack for LSML and BDA applications involving ACGs. To this end, we must work with real world problems such as cancer-related genetic biomarkers, textual topic analysis, and fraud detection in social networks.

### 2.5 Graph Generator for Synthetic Benchmarking

The graph generator must produce synthetic ACGs and network information from Big Data datasets that are suitable for benchmarking the exascale storage and I/O subsystem.

#### 2.5.1 [Mandatory] The graph generator must store synthesized graphs using the HDF5 data model

The graph generator must use HAL APIs to store synthesized real world graphs and network information using the HDF5 data model.

**The information on this page is subject to the use and disclosure restrictions provided on the cover page to this document.**
**Copyright 2014, Intel Corporation.**

### 2.5.2 [Mandatory] The graph generator must produce synthetic large-scale ACGs that have similar statistical properties to the corresponding real-world seed graphs

The graph generator must produce synthetic graphs based on the topological and statistical properties of real-world seed graphs. These seed graphs must be derived from Big Data datasets using the ACG ingress process.

### 2.5.3 [Mandatory] The graph generator must construct graphs using Hadoop MapReduce software and commercial-off-the-shelf hardware

The synthetic graph generator must construct graphs using a system that represents a conventional Big Data Analytics cluster computing system. It is a non-goal to research graph construction using the exascale architecture. Rather, graphs must be constructed on COTS systems and the output loaded into the exascale storage and I/O subsystem.

### 2.5.4 [Mandatory] The graph generator must attach representative network information to the synthesized graphs

The graph generator must be able to attach network information with representative real-world statistics (e.g., size distribution and type) to synthesized graphs using the HAL APIs. This network information may be synthesized from seed Big Data datasets. This requirement ensures that we will be able to realistically and exhaustively benchmark exascale storage and I/O load performance.

### 2.5.5 [Desirable] Run-time synthetic benchmarking

We may develop synthetic graphs and network information that are suitable for run-time benchmarking of the exascale storage and I/O subsystem. This would require that the synthesized information be appropriate for one or more of the structured machine learning or graph analytics algorithms that we develop. We would evaluate:

- Execution time for graph computation
- Run-time use of the storage and I/O subsystem for graph scheduling
- Memory used by a compute node during graph computation (to observe how the storage and I/O subsystem lowers the footprint)

**The information on this page is subject to the use and disclosure restrictions provided on the cover page to this document.**
**Copyright 2014, Intel Corporation.**

# 3 Use Cases

## 3.1 Choice of applications

Our applications use algorithms that perform asynchronous local updates on ACGs and iterate to convergence. Key graph-based machine learning and graph analytics algorithms may be broken down into the following categories:

- Clustering and Classification
- Collaborative Filtering
- Graphical Models
- Graph Analytics

We have selected applications that will utilize algorithms derived from two or more of these categories and that rely on Big Data datasets and may benefit from exascale supercomputers and the storage and I/O subsystem under study. These applications are:

- *De novo* identification of pathways in cancer progression
- Hidden topic analysis from large unstructured data corpuses
- Fraud and anomaly detection from text documents

The nature of the Big Data and HPC processing required for the above applications varies substantially from one to another. For example, the *de novo* pathways use-case involves making inferences based on graph-based search and optimization. The text mining application runs probabilistic sampling over a graphical model and involves key-value indexing operations. Our third application involves computing various neighborhood properties, and, possibly, spectral coordinates. We believe these applications represent potential future DOE target applications and will permit us to study the load and run-time performance of the exascale storage and I/O stack while motivating Big Data-HPC bridge usage models.

## 3.2 Canon: an application for the *de novo* identification of biological pathways in cancer progression

The process of making new discoveries in the molecular biology of cancer requires advanced computational modeling techniques which take into account the relationships between different stages in protein expression—from genome copy number variation, to gene expression, to its activity and final products. Despite advancements, the continued progression of the field is limited by two important factors: the quality of curated literature-bases, and limited computational resources. Resources made available from groups like the National Cancer Institute (NCI), the National Center for Biotechnology and Information (NCBI), and The Cancer Genome Atlas (TCGA) make it possible to use factor analytic-type methods to construct statistical models of changes in the information flow of cancer genomes, and their relationships to certain outcome variables, like cancer

**The information on this page is subject to the use and disclosure restrictions provided on the cover page to this document.**
**Copyright 2014, Intel Corporation.**

progression, or therapeutic response to certain drug treatments. Although such models have been effective, they are fundamentally limited by computational resources and the accuracy of the literature-base upon which they are curated. Ideally, computational models would only require the specific entities described in such databases (e.g., genes, proteins, etc.), rather than the entities and the relationships between them. This is not, however, feasible at this time, due to the second factor limiting the advancement of our knowledge of cancer cell biology—computational power. The absence of the above-described relational assumptions was recently proposed as desirable, yet computationally intractable in a recent publication [15].

Figure 1 depicts a simplified sub-pathway that results in the inhibition of apoptosis. At a macro level (left), MDM2 inhibits TP53, which increase apoptotic activity. These high-level, literature-curated relationships are a macrocosm of a series of known biological pathways describing the relationships between DNA, mRNA, Protein, and methylated (activated) protein (right). A simplifying assumption that has been a part of previous modeling approaches, such as PARADIGM [15], has been the high-level inhibitory/excitatory relationships depicted on the left-hand side of Figure 1. Considering that there are three possible relationships between two entities (inhibition, excitation, and no relationship), and that there are on the order of 5000 entities in the model space, the problem size suffers a quick combinatorial explosion; even the ideas of careful pruning remain beyond the computers commonly available to biologists.

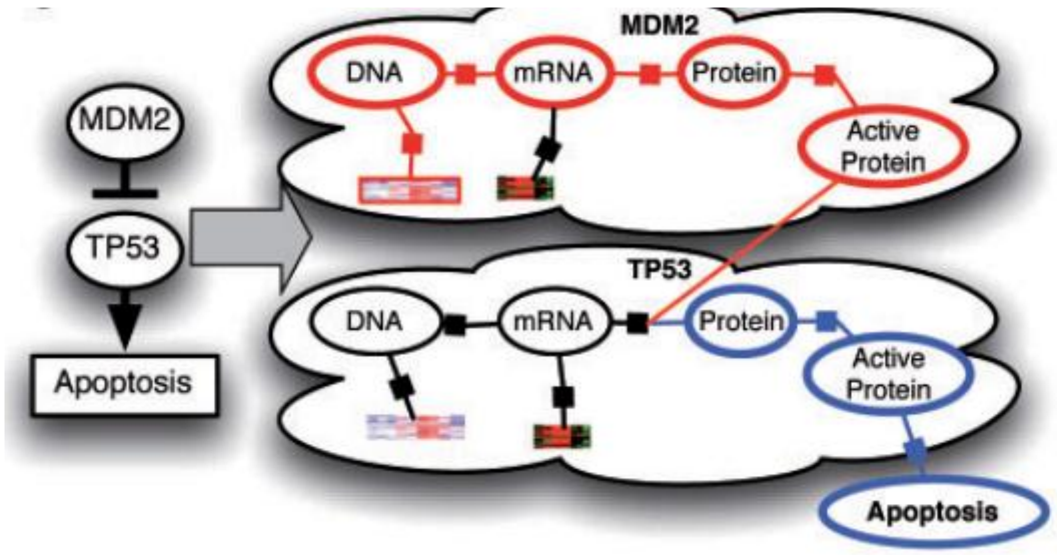

**Figure 1. Simplified biological sub-pathway resulting in the inhibition of apoptosis (cell death), and depicting certain relationships assumed from the Vaske et al., 2010 approach to modeling. Connections between the MDM2 "Active Protein" and the TP53 mRNA translating to "Protein" are both relationships that have been assumed from the curated literature. Other relationships (e.g., DNA to mRNA) are not a part of these assumptions, but they are part of the biological canon.**

**The information on this page is subject to the use and disclosure restrictions provided on the cover page to this document.
Copyright 2014, Intel Corporation.**

The primary goal of the proposed application is to re-engineer the PARADIGM algorithm to make no assumptions about the inhibitory/excitatory relationships between entities in biological pathways. This is an exciting, biologically-relevant, exascale problem that can only be approached using the presently-described architecture. We will explore, with appropriate branch-and-bound guidelines, the vast configuration space of the combinations of relationships between entities (including, for modeling purposes, an "Anonymous" entity, to signify an as-yet-unknown node in the pathway), distribute them over our exascale test bed, and use structured (graph-based) analogues of optimization algorithms (e.g., Structured Expectation Maximization, or CO-EM) to search the graph space for the pathway(s) that best explain(s) observed data regarding cancer outcomes, such as therapeutic responses, and rate of disease progression. We call the system *Canon*, because a side-effect of our approach will be the ability to statistically validate the curated literature, in terms of inhibitory/excitatory relationships between biological entities in the cancer cell. If a previously-reported relationship is also included in a model geared toward finding the statistically best-possible pathway in a system, then it is even more likely to be real, rather than an experimental anomaly; it is canon.

As a first step, we propose to independently implement the PARADIGM algorithm described by Vaske and colleagues [15], and cast it into a graph representation. This problem on its own, while difficult, does not necessitate exascale-capable systems. It will, however, give us the opportunity to verify the handling of networked data by our system against an already-established algorithm. Once we are able to verify our results for this, we will remove the assumptions regarding the inhibitory/excitatory relationships between nodes in the macro-scale biological pathways, and directly engage the exascale formulation of this problem.

Our approach will demonstrate significant advances to both the ML and bioinformatics research communities. In terms of ML, our system will show how graph-parallel ML algorithms can be used on real-world exascale data to solve problems that would not be tractable without representing the data structures involved as ACGs, and running the optimization algorithms over an asynchronous computational kernel with access to the proposed I/O subsystem. For the bioinformatics community, we show that extending a state-of-the-art cancer genomics algorithm to perform in an assumption-agnostic manner can lead to novel results. We are confident in this because there are two possible outcomes to our experiments. One possibility is that our system will be able to discover previously-unknown relationships between information contained in the breast cancer genome and therapeutic outcomes. The other possibility is that our system shows that the previous formulations of the probabilistic graphical model-based approach were sufficiently accurate, despite the simplifying assumptions they

**The information on this page is subject to the use and disclosure restrictions provided on the cover page to this document.**
**Copyright 2014, Intel Corporation.**

needed to make. This would be a valuable, publication-worthy result for the bioinformatics community, as it would constitute quantitative evidence that their simplifying assumptions did not change the results of their experiments, the practical upshot of which would be that not every lab would need to invest in a high performance computing system.

Aside from its importance to the bioinformatics and cancer biology communities, this problem will be able to uniquely test the scalability of the optimization algorithms we adapt for exascale searches over the graph-space for the path that maximizes the likelihood of the outcome-level observations. Expectation Maximization (EM), for example, is a commonly-used algorithm for finding the maximum likelihood estimates of the parameters in a statistical model, when that model involves unobserved, or latent, variables, as ours does in the above-described use case. This is a situation that is frequently encountered in the machine learning community, so observing how structured variants of this approach (e.g., Structured EM, CO-EM) scale to our architecture will be important for other use cases that end users may encounter in the future. An important contribution of our approach will be methods for handling exascale, distributed data. Although any particular network will easily fit in memory, the collection of all possible networks will need to be distributed amongst computing nodes in such a way that our ML algorithms can efficiently mine their data and will rely on an object-based storage architecture.

## 3.3 Topic modeling with Latent Dirichlet Allocation

Enabling a computer to understand and draw inferences from textual data has a variety of useful applications. This was demonstrated on a small scale in 2010 when Watson, a supercomputer from IBM based on a complex AI architecture known as DeepQA, defeated the then-undefeated jeopardy champions on the national television. Amongst the wide range of applications for an exascale supercomputer is the possibility of a sophisticated text mining aid for researchers (especially in medicine and bioinformatics) that are in constant need of searching through vast unstructured corpora.

The complex architecture of a text mining system, such as that of DeepQA, typically comprises a sequence of functional modules [16]. Since its raw input is always unstructured or semi-structured, any such sequence begins with modules that can parse and extract different structures out of such a corpus. Topic modeling is one such application, used for discovering topics latent inside a document collection, which is sometimes a useful tool at the start of a multi-stage text-mining workflow.

Topic modeling may be performed using a graph-based representation of text, and then running modern statistical algorithms, such as Latent Dirichlet Allocation

**The information on this page is subject to the use and disclosure restrictions provided on the cover page to this document. Copyright 2014, Intel Corporation.**

(LDA) over the corresponding graphical model. Such a graph can be prohibitively large (Modeling) [17] since its size is typically a low-degree polynomial of the size of its raw input. For example, MEDLINE, the bibliographic database maintained by the U.S. National Library of Medicine (NLM) for references to journal articles in the health sciences, contains over 19 million references at present [18]. The website estimates that a new reference gets added to this database every two minutes. Similarly, the corpora of scholarly medical texts across the world libraries is estimated to have trillions of pages of texts, and the researchers in medicine and bioinformatics are increasingly interested in text mining-based solutions for extracting higher-order meaningful information (e.g., relating a specific genetic polymorphism to a disease) from that massive corpus [19]. Over a corpus of that size and a realistic set of few hundred thousand possible topics, a bipartite graph has $\boldsymbol{O}(10^{17})$ edges and even a linear algorithm on that input involves many sextillions—$\boldsymbol{O}(10^{21})$—of operations. Topic modeling has many other potential applications. For example, identifying and flagging on evolving topics could form the basis of an email-network monitoring system in the context of anti-terrorism.

We propose to use LDA-based topic modeling as our second use case.
For the dataset, we will use publicly-available resources (e.g., the Wikipedia corpus), and, if needed, scale them up to sizes suitable for the storage and I/O subsystem demonstration. We will pre-process the dataset using the ACG ingress process and extract the underlying graphical structures with associated network information. The generated graphs will be partitioned and loaded into the exascale storage and I/O subsystem from a Hadoop MapReduce cluster. We will implement a statistical sampling algorithm for extracting the latent topics on the system under test. There are multiple candidate algorithms to choose from. As a starting point, we will experiment with a variant of Gibbs sampling that is based on Markov-chain Monte Carlo principle. The sampling techniques assume that there is a Dirichlet prior distribution latent in how the topics are embedded in the documents. This is a powerful and widely-used modern approach, and we are interested in studying its object access patterns and storage requirements.

## 3.4 Fraud and anomaly detection from text documents

Fraud and anomaly detection is needed in a variety of contexts, including social networks, online business, national security, airline safety, and network security. Many organizations, both government and private, are interested in mining information about threat, anomaly, and dishonesty. For example,

- Online service providers (and their consumers) are showing interest in isolating illegitimate users [20] whose malicious activities include spamming, propagating malware, creating biased product reviews and other forms of social engineering-type attacks.

**The information on this page is subject to the use and disclosure restrictions provided on the cover page to this document.**
**Copyright 2014, Intel Corporation.**

- Large organizations, and the proprietors of P2P networks want to detect and isolate infiltrating botnets that can launch different kinds of attacks, such as distributed denial of service, spamming, phishing, or other criminal attacks [21].
- NASA's aviation safety program is interested in quickly isolating safety violations by analyzing anomalies and outliers from various available data sources, such as maintenance logs, air traffic reports, reports from airline personnel, and even aircraft black boxes [22].
- Various large organizations are interested in identifying different threats, such as malicious financial transactions, intellectual property-theft, and are turning to automated tools to help carry out these tasks [23]. Similarly, the Department of Homeland Security has been using data mining tools to catch biological threats [22].

We will run a set of anomaly detection algorithms, such as those proposed in [24] or [25] that are useful for detecting biased reviews in online marketplaces or for detecting frauds. Today, the size of graphs that arise out of such networks easily scale up to quadrillions [26, 25]. For the data, we will choose from publicly-available datasets, such as the Common Crawl Corpus or Material Safety Data Sheets located at http://aws.amazon.com/datasets, and scale them appropriately for the prototype hardware or generate artificial datasets of our own using the proposed graph generator. We will investigate subsets of vertices in the resultant graphs that have anomalous connectivity structures. These anomalies may not be readily observed to the naked eye – as would in- or out-degree patterns – but may stand out in subtle ways, such as in spectral coordinates. Our studies will reveal how these types of computations exercise the storage and I/O stack.

## 3.5 Synthetic Benchmarking of Load and Run-Time Performance

The benchmarking of the exascale storage and I/O subsystem requires algorithms and ACGs that will utilize and stress its new object-based features. Generally speaking, obtaining real-world data that is sufficiently large and complex enough for testing an exascale supercomputer is challenging. And, it is particularly difficult to find datasets that are suitable for extensive performance characterization of a system. We aim to address this challenge by generating large-scale synthetic graph structures and network information that we can statistically control for performance sweeps and corners testing. We believe that a synthetic graph generator will allow us to benchmark the storage and I/O subsystem more completely, and detect performance bottlenecks faster, than real-world datasets and applications will allow. We will investigate techniques for synthesizing ACGs that emulate graphs arising from real-world Big Data problems. Our goal is to synthesize ACGs with the topological and statistical properties that reflect those of

**The information on this page is subject to the use and disclosure restrictions provided on the cover page to this document.**
**Copyright 2014, Intel Corporation.**

real world graphs (for example, power law degree distribution, preferential attachment, small diameter, short average path length, community structures, and size scalability) [27, 28]. For real world graph models, we are interested in power-law graphs, in which the degree distribution is characterized by a rapidly-decaying power-law tail. Recent work has confirmed that the power-law graph model effectively captures the aforementioned properties of real-world networks [29].

We will use an extension of the Kronecker multiplication method proposed by Leskovec et al. to generate graphs [30]. Leskovec showed that the Kronecker graph generation algorithm can synthesize large graphs with similar topological and statistical properties to small seed graphs. The Kronecker graph generation algorithm has been adopted by the Graph500, the data-intensive supercomputing benchmark project. We will extend this work by (1) introducing the stochastic Kronecker graph generation algorithm and (2) investigating whether stochastic Kronecker graphs can scale up to exascale. These synthetic graphs will be partitioned into multiple graph components for efficient computation. In addition, we will next investigate synthesizing large quantities of network information from smaller Big Data sources and associating it with synthesized graphs. Finally, BDA or LSML applications will execute graph computations over the synthetic data graphs in carefully-controlled I/O benchmarking experiments.

**The information on this page is subject to the use and disclosure restrictions provided on the cover page to this document.**
**Copyright 2014, Intel Corporation.**

# 4 Solution Proposal

Our primary objective is to evaluate the load and run-time performance of the exascale storage and I/O stack for LSML and BDA workloads. In this section, we describe the solution architecture for the ACG ingress process, the computational kernel that will exercise the subsystem at run-time, our proposed test applications, and a synthetic benchmark. We propose to use GraphBuilder open source software as the basis for the ACG ingress process and GraphLab as the basis for the computational kernel. Both will need to be adapted to the HDF5 API.

For GraphBuilder, we will need to:

- Obtain and develop new input datasets
- Write new parsing and extraction programs
- Add new features and capabilities to the library
- Explore new graph partitioning algorithms
- Adapt its output to the HDF5 data model by using current or expanded HDF5 data types that will be provided in the HDF5 APIs

For GraphLab, we will need to:

- Write new vertex programs
- Add new features and capabilities to the application framework
- Develop a run-time storage architecture that uses the current or expanded HDF5 data types that will be provided in the HDF5 APIs
- Modify the computational kernel to try offloading some of the data consistency, synchronization, and function update mechanisms to the exascale storage and I/O subsystem

## 4.1 Background

### 4.1.1 GraphBuilder and GraphLab

**GraphBuilder**: GraphBuilder, developed at Intel Labs [14] is an open source (Apache 2.0) scalable graph construction library for Hadoop MapReduce that provides a simple Java library with algorithms for parallel graph construction, transformation, and verification (Figure 2). The self-describing JSON output is suitable for large-scale graph mining. Today, GraphBuilder stores output graphs to the Hadoop Distributed File System (HDFS). GraphBuilder may also partition and serialize large-scale graphs for use by GraphLab (see below) and other parallel ML frameworks. Today, each graph partition is stored as a separate file in HDFS. We will use this library as the ACG ingress process for creating graph structures with associated network information out of large unstructured datasets.

**The information on this page is subject to the use and disclosure restrictions provided on the cover page to this document.**
**Copyright 2014, Intel Corporation.**

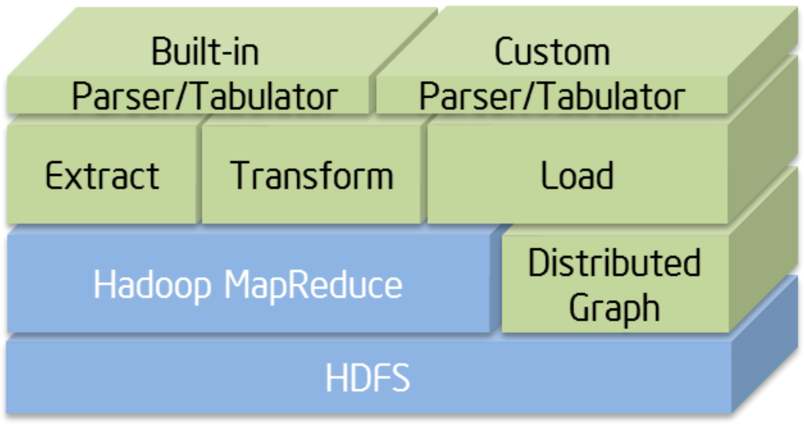

**Figure 2. GraphBuilder software stack**

**GraphLab:** GraphLab is a scalable parallel ML framework, developed at Carnegie Melon and University of Washington [11, 12]; it has been released as an open source software library under Apache 2.0 licensing. GraphLab encapsulates a set of salient systems-level features that are repeatedly used by many parallel algorithms in ML and data mining. It has four major architectural ingredients:

- Graph representation and ability to send (and receive) asynchronous updates
- Shareable data-table for global objects
- A scheduling feature that allows users to sequence graph-based operations
- Synchronization primitives that let users specify different degrees of data consistency

GraphLab implements a *Gather-Apply-Scatter* (GAS) computational model to distribute intermediate updates asynchronously between independent vertex programs. GraphLab's features make it a convenient high-performance computational kernel for LSML and BDA algorithms and applications.

Figure 3 depicts GraphBuilder and GraphLab in the context of Big Data-HPC bridge. The software libraries will communicate with the exascale storage and I/O subsystem using the HAL. We will use HDF5 to access stored graphs and network information, for efficient data-synchronization and buffering, and for pushing more advanced data transformation programs to the storage and I/O subsystem.

**The information on this page is subject to the use and disclosure restrictions provided on the cover page to this document.**
**Copyright 2014, Intel Corporation.**

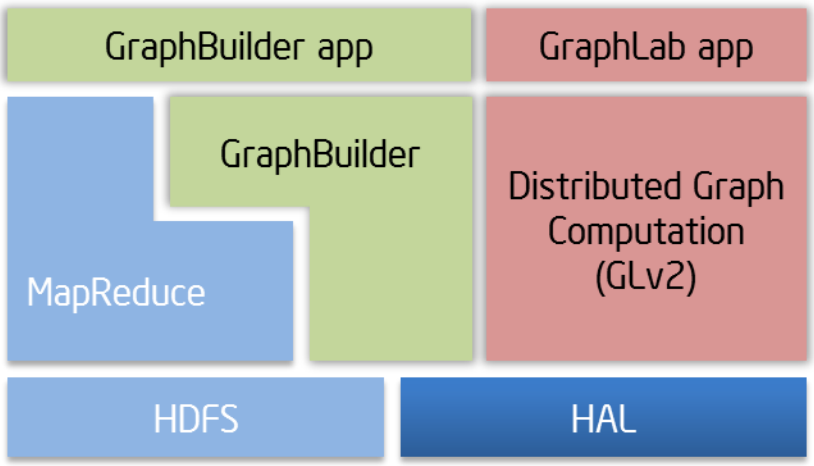

**Figure 3. Big Data-HPC bridge software stack.**

### 4.2 Develop HDF5 adaptation layer for Graph Representation

We will use an HDF5 pointer data type (co-developed with other members of this project) to efficiently translate representations of the graph data to and from HDF5. An HDF5 Abstraction Layer will be used to store both graphs and associated network information. We will also investigate non-pointer graph representations, such as matrices and tensors and optimal techniques for conversion between these different formats, and seek the best structures and methods for storing graphical information. In addition to representation, we will also experiment with data structures to best utilize the HDF5 datasets and meta-data for to minimize communication over the I/O stack. In summary, this layer will satisfy the requirements outlined in Requirement 2.1.

### 4.3 Adapt and extend capabilities for ACG ingress

In order to meet Requirement 2.2, *i.e.,* the ACG ingress process, we will modify the GraphBuilder library. Here, the mandatory change is replacing its output storage interface based on JSON/HDFS with calls to the HAL that will expose the exascale storage and I/O stack to these libraries.

We will add an extension module to GraphBuilder that will convert the standard output graph format (e.g., in adjacency list format) to the HDF5-based representation outlined in Section 4.2. The changes will be made to incorporate both the underlying graph, as well as the associated network information. We will also consider additional improvements to this library, such as advanced partitioning strategies, discussed in Section 4.5.

**The information on this page is subject to the use and disclosure restrictions provided on the cover page to this document.**
**Copyright 2014, Intel Corporation.**

## 4.4 Develop or extend the capabilities of the computational kernel

### 4.4.1 Use or extend asynchronous I/O communication

GraphLab's vertex programs update each other asynchronously using compute-compute network communications. We will use this as a baseline to satisfy Requirement 2.3.1. We will investigate if there is any advantage to sharing vertex and/or edge updates through new mechanisms provided by the exascale storage and I/O architecture subsystem.

### 4.4.2 Use or extend data sharing feature

Data sharing is an important aspect of our computational kernel (see Requirement 2.3.2), and GraphLab already provides this feature. As baseline, we will use this built-in support. However, we will investigate whether there is any advantage to sharing data structures through new mechanisms provided by the exascale storage and I/O subsystem.

### 4.4.3 Efficient representation of graph partitions and sub-partitions in HDF5 data model

We will work with members of other subteams to develop a representation of graph partitions and sub-partitions for HDF5. A means of representing partitions so that the updates, the merging of components and meta-data modifications do not generate significant overhead, is an open research question. We propose to experiment with different graphs to examine how partitions can be best represented and indexed.

### 4.4.4 Use of analysis shipper

In response to Requirement 2.3.5, we will investigate the use of the analysis shipper to improve execution efficiency and lower I/O throughput demand by selectively shipping analyses closer to where data is warehoused via the analysis shipper. Our initial plan is to investigate using the analysis shipper for feature extraction, subgraph retrieval, and simple data transforms with large input to output information ratios. However, this investigation is conditioned upon the feasibility of invoking the analysis shipper on-demand from a compute-node.

### 4.4.5 Indexing and prefetching of graph sub-partitions

In response to Requirement 2.3.3 and in conjunction with the indexing feature to be implemented in the HDF5 adaptation layer, we will implement a sequencing feature to efficiently schedule and access subsets of HDF5 objects (such as graph sub-partitions) at run time. This will mimic the parallel sliding window style of processing used by other researchers [31] for handling multiple partitions on a single node. We expect significant performance improvement by pre-sequencing such fetches in conjunction with efficient indexing.

**The information on this page is subject to the use and disclosure restrictions provided on the cover page to this document.
Copyright 2014, Intel Corporation.**

### 4.4.6 Advanced streamlining of data access through data usage hints

For situations outlined in Section 2.3.4, we will create APIs in the HAL to call HDF5 APIs with appropriate parameters signaling data usage hints. The HDF5 library will take these hints and implement appropriate mechanisms to influence the behavior of the lower levels in the stack. These operations, although hidden from the application, will improve their I/O efficiency by streamlining their data access during the load-compute-unload cycle. We expect to uncover data-usage patterns as our experiments progress. If we cannot identify such static or predictable data-usage patterns then we will not support this feature.

### 4.4.7 Use of transactional I/O

Although the computational kernel already provides consistency mechanisms to prevent write-write races, we may investigate using the transactional I/O feature (Requirement 2.3.6) for this purpose. Race prevention may be necessary to ensure algorithmic correctness and rapid convergence.

### 4.4.8 Bundling in burst buffers

When a compute node generates a large number of parallel asynchronous updates *(*Requirement 2.3.6*)* in a relatively short period of time*,* we may attempt to lower the number of I/O operations by bundling independent transactions together and holding them at the burst buffer before flushing (via the HDF5 API) them to storage.

## 4.5 Algorithmic questions and extension modules for GraphBuilder and GraphLab

### 4.5.1 Graph partitioning for Big Data graph computations

Improving the quality of vertex cuts through a graph lowers the number of data dependencies between partitions and minimizes network communications. At the same time, the compute effort must be load balanced by place approximately the same number of edges on each compute node. In our solution, we will start with GraphBuilder's existing partitioning heuristics, i.e., random partitioning and an oblivious greedy algorithm. We may investigate implementing new partitioning strategies in GraphBuilder, especially adaptive ones. For example, we will investigate how we can best leverage near linear-time crude-partitioning strategies for ACG applications; these techniques are otherwise known to produce good results [32] but to the best of our knowledge, have not been applied to ACGs.

### 4.5.2 Additional techniques, such as graph-spectral methods

Some of the application classes identified will benefit a great deal from using graph-spectral techniques. However, it is easier to implement these algorithms if the graphs are represented differently, such as with matrices or tensors [32, 26,

**The information on this page is subject to the use and disclosure restrictions provided on the cover page to this document.**
**Copyright 2014, Intel Corporation.**

33, 24, 25]. Many algorithmic questions reduce to finding the top eigenvalues (say, in a web graph), or creating a tensor-factorization (in a belief network).

In our work, we wish to answer the following research questions:

- How can we best implement the data structures so that switching between different graph representations is efficient?
- Can different types of data representations offer better retrieve/store performance? We believe that in such cases the contiguous nature of the data can be used to our advantage, in conjunction with direct HDF5 features such as hyperslab extraction.

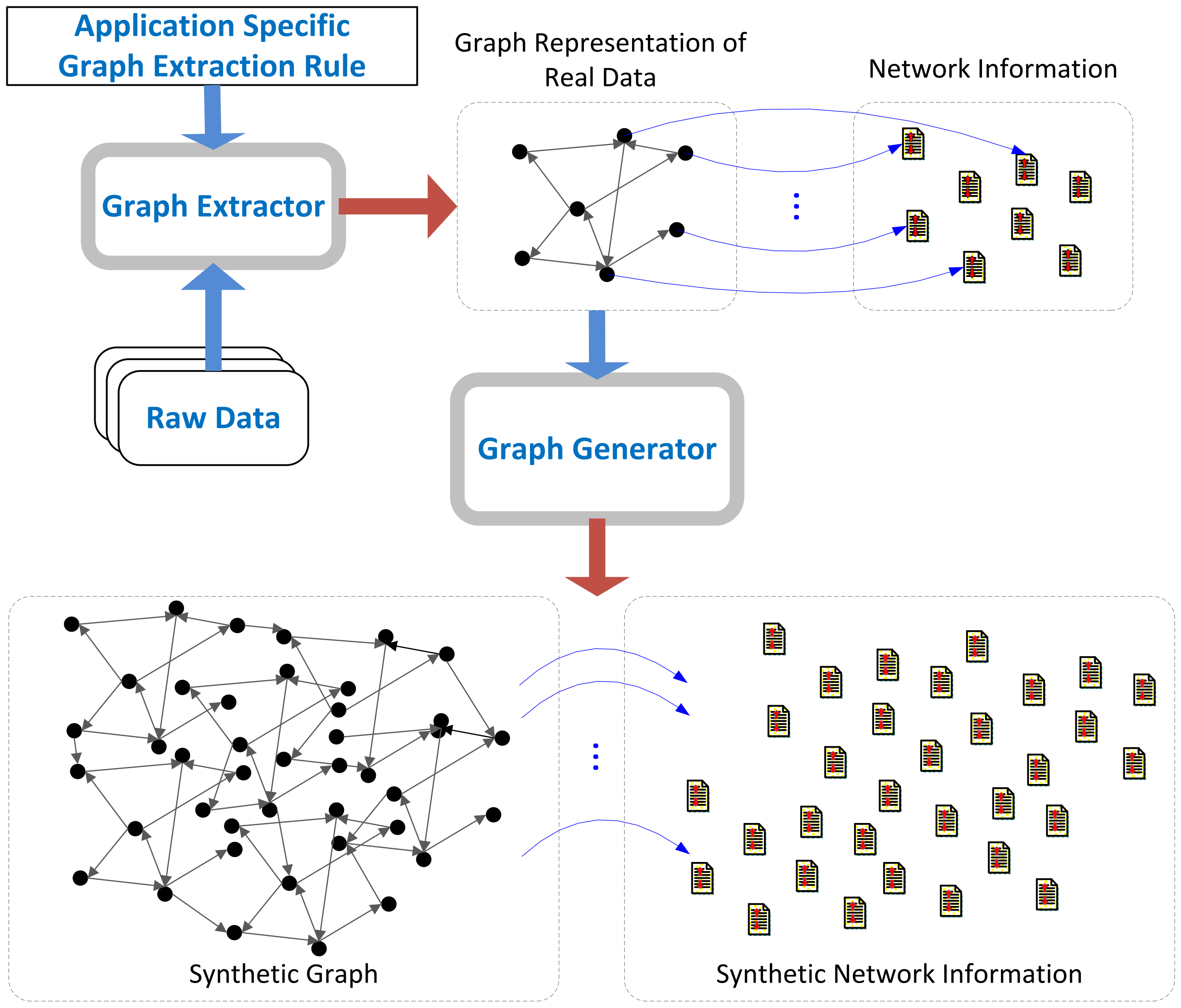
Application Specific Graph Extraction Rule
Graph Extractor
Raw Data
Graph Representation of Real Data
Network Information
Graph Generator
Synthetic Graph
Synthetic Network Information

**Figure 4. Workflow for constructing synthetic Big Data graphs.**

## 4.6 Application testing

We will investigate the suitability of the exascale storage and I/O stack for LSML and BDA applications involving ACGs. To this end, we will work with three categories of graph problems found in different real world settings: cancer-related genetic biomarkers, textual topic analysis, and fraud detection in social networks. We will test at least one, but intend to test all three.

## 4.7 Graph Generator for Synthetic Benchmarking of Exascale Storage and I/O Subsystem

The graph generator will synthesize arbitrarily-large ACG datasets to benchmark the performance of the exascale I/O storage subsystem for LSML and BDA applications. This section describes the proposed solutions to the requirements outlined in Section 2.5

### 4.7.1 Graph representation in HDF5 data model

The synthetic graphs will be represented in HDF5 data model using HAL APIs.

### 4.7.2 Graph generation algorithm

We will use the (stochastic) Kronecker multiplication method to generate synthetic real world graphs (i.e., Kronecker graphs). Although it is known that the stochastic Kronecker graphs can scale up to more than a million nodes without losing the properties of their associated seed graphs, we will further increase the size of the graphs to exascale to meet our needs.

### 4.7.3 Exascale graph synthesis

Our exascale graph generator will run on a conventional Hadoop MapReduce compute cluster.  Intermediate data will be temporarily stored in HDFS in this system. The final output will be converted into the HDF5 data model and stored into the exascale storage and I/O subsystem.

### 4.7.4 Data annotation on the synthetic graphs

To attach network information to synthesized graphs, we will implement two different methods: (1) reusing copies of real data as many times as needed and (2) using artificially generated data.

### 4.7.5 Application level synthetic benchmarking

For application-level synthetic benchmarking of the ingress and run-time performance of the storage and I/O subsystem, we will use the ACGs and network information that will be synthesized by our Kronecker graph generator and ACG ingress process. During computation, the synthetic ACGs will produce I/O access patterns and loading that will emulate those of real data. To evaluate the

**The information on this page is subject to the use and disclosure restrictions provided on the cover page to this document.**
**Copyright 2014, Intel Corporation.**

application-level performance of ingress and run-time application I/O performance, we will develop quantitative and/or qualitative performance metrics.

**The information on this page is subject to the use and disclosure restrictions provided on the cover page to this document.**
**Copyright 2014, Intel Corporation.**

# 5 Acceptance Criteria

## 5.1 HDF5 Adaptation Layer

- GraphLab, GraphBuilder, or any application that uses HDF5 adaptation layer APIs can successfully create, access, and remove HDF5 objects (graphs, subgraphs, and other ACG-related data-objects).
- HDF5 *efficiently* represents ACGs.

## 5.2 GraphBuilder Modifications

- Successfully store graphs in HDF5 data model and create correct partitions of graphs on specified algorithms.

## 5.3 Ingress data access to I/O subsystem

- Successfully store graphs and network information into the storage and I/O subsystem.
- Successfully store graph partitioning, sharing, and other data locality information.

## 5.4 Runtime data access via I/O subsystem

- Compute nodes successfully load, compute, asynchronously update, and unload partitioned graphs and associated data.

## 5.5 Big Data graph generation

- Successfully synthesize graphs that mimic real world graph topologies and statistics.
- Demonstrate that the solution is likely to scale to exascale for real-world graph construction.
- Successfully associate the synthesized graphs with network information.

**The information on this page is subject to the use and disclosure restrictions provided on the cover page to this document.**
**Copyright 2014, Intel Corporation.**

## Reference

[1] "Big Data," 11 1 2013. [Online]. Available: http://en.wikipedia.org/wiki/Big_data.

[2] "Machine Learning," Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/Machine_learning. [Accessed 10th January 2013].

[3] D. J. Watts and S. H. Strogatz, "Collective Dynamics of "Small-world" networks," *Nature,* vol. 393, no. 6684, pp. 440-442, 1998.

[4] M. Kim and J. Leskovec, "Modeling Social Networks with Node Attributes using the Multiplicative Attribute Graph Model," *arXiv preprint arXiv:1106.5053,* 2011.

[5] M. Faloutsos, P. Faloutsos and C. Faloutsos, "On power-law relationships of the internet topology," *ACM SIGCOMM Computer Communication Review,* vol. 29, no. 4, pp. 251-262, 1999.

[6] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins and J. Wiener, "Graph Structure in the Web," *Computer Networks,* vol. 33, no. 1, pp. 309-320, 2000.

[7] H. Ebel, L.-I. Mielsch and S. Bornholdt, "Scale-free topology of e-mail networks," *Physical Review E,* vol. 66, no. 3, p. 35103, 2002.

[8] R. F. i Cancho and R. V. Solé, "The small-world of human language," *Proceedings of the Royal Society of London. Series B: Biological Sciences,* vol. 268, no. 1482, pp. 2261-2265, 2001.

[9] K. Andreev and H. Racke, "Balanced graph partitioning," *Theory of Computing Systems,* vol. 39, no. 6, pp. 929-939, 2006.

[10] A. Abou-Rjeili and G. Karypis, "Multilevel algorithms for partitioning power-law graphs," in *Parallel and Distributed Processing Symposium*, Rhodes Islands, Greece, 2006.

[11] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin and J. M. Hellerstein, "Graphlab: A new framework for parallel machine learning," *arXiv preprint arXiv:1006.4990,* 2010.

[12] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin and J. M. Hellerstein, "Distributed GraphLab: A Framework for Machine Learning and Data Mining in the Cloud," *VLDB Endowment,* vol. 5, no. 8, pp. 716-727, 2012.

[13] "The Graph 500 List," 2010. [Online]. Available: http://www.graph500.org/. [Accessed 10 January 2013].

[14] T. L. Willke, N. Jain and H. Gu, "GraphBuilder – A Scalable Graph Construction Library for Apache™ Hadoop™," in *Neural Information Processing Systems*, Lake Tahoe; NV, 2012.

**The information on this page is subject to the use and disclosure restrictions provided on the cover page to this document.**
**Copyright 2014, Intel Corporation.**

[15] C. J. Vaske, S. C. Benz, J. Z. Sanborn, D. Earl, C. Szeto, J. Zhu, D. Haussler and J. M. Stuart, "Inference of patient-specific pathway activities from multi-dimensional cancer genomics data using PARADIGM," *Bioinformatics,* vol. 26, no. 12, pp. i237-i245, 2010.

[16] D. Ferrucci, E. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. A. Kalyanpur, A. Lally, J. W. Murdock, E. Nyberg, J. Prager, N. Schlaefer and C. Welty, "Building Watson: An overview of the DeepQA project," *AI Magazine,* vol. 31, no. 3, pp. 59-79, 2010.

[17] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton and T. Vassilakis, "Dremel: interactive analysis of web-scale datasets," *VLDB Endowment,* vol. 3, no. 1-2, pp. 330-339, 2010.

[18] "Fact Sheet - MEDLINE®," U.S. National Library of Medicine, [Online]. Available: http://www.nlm.nih.gov/pubs/factsheets/medline.html. [Accessed 10th January 2013].

[19] R. L. Grossman and K. P. White, "A vision for a biomedical cloud," *Journal of internal medicine,* vol. 271, no. 2, pp. 122-130, 2012.

[20] Y. Xie, F. Yu, Q. Ke, M. Abadi, E. Gillum, K. Vitaldevaria, J. Walter, J. Huang and Z. M. Mao, "Innocent by association: early recognition of legitimate users," in *ACM conference on Computer and communications security*, New York, NY, USA, 2012.

[21] S. Nagaraja, P. Mittal, C.-Y. Hong, M. Caesar and N. Borisov, "BotGrep: finding P2P bots with structured graph analysis," in *Proceedings of the 19th USENIX conference on Security*, Washington, DC, 2010.

[22] P. Marshall, "NASA applies deep-diving text analytics to airline safety," 26th October 2012. [Online]. Available: http://gcn.com/articles/2012/10/26/nasa-applies-text-analytics-to-airline-safety.aspx. [Accessed 10th January 2013].

[23] Y. Kim and F. Sheldon, "Anomaly detection in multiple scale for insider threat analysis," in *Proceedings of the Seventh Annual Workshop on Cyber Security and Information Intelligence Research*, 2011.

[24] X. Ying, X. Wu and D. Barbará, "Spectrum based fraud detection in social networks," in *IEEE 27th International Conference on Data Engineering*, Hannover, Germany, 2011.

[25] U. Kang, "Mining Tera-Scale Graphs: Theory, Engineering and Discoveries," Pittsburgh, 2012.

[26] U. Kang, C. E. Tsourakakis and C. Faloutsos, "Pegasus: A peta-scale graph mining system implementation and observations," in *Ninth IEEE International Conference on Data Mining*, 2009.

[27] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science,* vol. 286, no. 5439, pp. 509-512, 1999.

[28] M. Girvan and M. E. J. Newman, "Community structure in social and

**The information on this page is subject to the use and disclosure restrictions provided on the cover page to this document.**
**Copyright 2014, Intel Corporation.**

biological networks," *Proceedings of the National Academy of Sciences,* vol. 99, no. 12, pp. 7821-7826, 2002.

[29] A. Clauset, C. R. Shalizi and M. E. Newman, "Power-law distributions in empirical data," *SIAM Review,* vol. 51, no. 4, pp. 661-703, 2009.

[30] J. Leskovec, D. Chakrabart, J. Kleinberg and C. Faloutsos, "Realistic, mathematically tractable graph generation and evolution, using kronecker multiplication," *Knowledge Discovery in Databases,* pp. 133-145, 2005.

[31] A. Kyrola, G. Blelloch and C. Guestrin, "GraphChi: Large-scale graph computation on just a PC," in *Proceedings of OSDI*, 2012.

[32] D. A. Spielman and S.-H. Teng, "Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems," in *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, 2004.

[33] D. A. Spielman and S.-H. Teng, "Spectral sparsification of graphs," *SIAM Journal on Computing,* vol. 40, no. 4, pp. 981-1025, 2011.

[34] S. L. Carter, C. M. Brechbühler, M. Griffin and A. T. Bond, "Gene co-expression network topology provides a framework for molecular characterization of cellular state," *Bioinformatics,* vol. 20, no. 14, pp. 2242-2250, 2004.

[35] A. Anagnostopoulos, R. Kumar, M. Mahdian, E. Upfal and F. Vandin, "Algorithms on Evolving Graphs," in *Innovations in Theoretical Computer Science Conference*, New York, NY, USA, 2012.

[36] B. Bahmani, R. Kumar, M. Mahdian and E. Upfal, "PageRank on an Evolving Graph," in *ACM SIGKDD international conference on Knowledge discovery and data mining*, New York, NY, USA, 2012.

[37] N. Friedman and M. Goldszmidt, "Sequential update of Bayesian network structure," in *Uncertainty in artificial intelligence*, San Francisco, CA, USA, 1997.

[38] J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos and Z. Ghahramani, "Kronecker graphs: An approach to modeling networks," *The Journal of Machine Learning Research,* vol. 11, pp. 985-1042, 2010.

[39] S. Ng, E. A. Collisson, A. Sokolov, T. Goldstein, A. Gonzalez-Perez, N. Lopez-Bigas, C. Benz, D. Haussler and J. M. Stuart, "PARADIGM-SHIFT predicts the function of mutations in multiple cancers using pathway impact analysis," *Bioinformatics,* vol. 28, no. 18, pp. i640-i646, 2012.

[40] M. Stratton, P. Campbell and P. Futreal, "The cancer genome," *Nature,* vol. 458, no. 7239, pp. 719-724, 2009.

[41] B. Zhang and S. Horvath, "A general framework for weighted gene co-expression network analysis," *Statistical applications in genetics and molecular biology,* vol. 4, no. 1, p. 1128, 2005.

**The information on this page is subject to the use and disclosure restrictions provided on the cover page to this document.**
**Copyright 2014, Intel Corporation.**

[42] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson and C. Guestrin, "PowerGraph: Distributed graph-parallel computation on natural graphs," in *Proceedings of the 10th USENIX Symposium onOperating Systems Design and Implementation (OSDI )*, 2012.

[43] C. L. Vogel, M. A. Cobleigh, D. Tripathy, J. C. Gutheil, L. N. Harris, L. Fehrenbacher, D. J. Slamon, M. Murphy, W. F. Novotny, M. Burchmore, S. Shak, S. J. Stewart and M. Press, "Efficacy and safety of trastuzumab as a single agent in first-line treatment of HER2-overexpressing metastatic breast cancer," *Journal of Clincal Oncology,* vol. 20, no. 3, pp. 719-726, 2002.

[44] S. F. Chin, A. Teschendorff, J. Marioni, Y. Wang, N. Barbosa-Morais, N. Thorne, J. Costa, S. Pinder, M. A. v. d. Wiel, A. Green, I. Ellis, P. Porter, S. Tavaré, J. Brenton, B. Ylstra and C. Caldas, "High-resolution aCGH and expression profiling identifies a novel genomic subtype of ER negative breast cancer," *Genome Biology,* vol. 8, no. 10, p. R215, 2007.

[45] N. Memon, J. D. Farley, D. L. Hicks, T. Rosenorn and (Eds.), Mathematical methods in counterterrorism, Springer, 2009.

[46] N. Memon, J. J. Xu, D. L. Hicks, H. Chen and (Eds.), Data mining for social network data, Springer, 2010.

[47] R. Sullivan, Introduction to data mining for the life sciences, Humana Press, 2011.

**The information on this page is subject to the use and disclosure restrictions provided on the cover page to this document.**
**Copyright 2014, Intel Corporation.**

**The information on this page is subject to the use and disclosure restrictions provided on the cover page to this document.
Copyright 2014, Intel Corporation.**