HIGH PERFORMANCE DATA ANALYTICS:

Experiences Porting the Apache Hama Graph Analytics Framework to an HPC InfiniBand Connected Cluster

1. Summary

Open source analytic frameworks, such as those in the Apache eco-system, provide access to large amounts of data in a productive and fault resilient way on scale-out commodity hardware systems. The objectives of High Performance Data Analytic (HPDA) systems are to maintain the productivity of the frameworks as well as to improve the performance for the Data Analyst. However, in order to achieve the performance available from High Performance Computing (HPC) technology, a framework must be recast from the distributed programming model used ubiquitously in the open source world to the parallel programming model used successfully in the HPC world. This “framework as an application” view includes the surgical replacement of key functions in the framework to leverage the strengths of HPC systems. We demonstrate this concept by porting the Apache Hama graph analytic framework to an HPC InfiniBand Cluster. By replacing the distributed barrier and message passing classes in the framework with HPC variants (prototyped in MPI), we achieved a performance increase of over 10x on a real-world Community Detection application applied to an 8 million vertice graph derived from Twitter data.

2. Motivation: High Performance Data Analytics

The development of open source analytic frameworks has provided a productive interface to tap the information content of vast amounts of data. These frameworks abstract the manipulation of the raw unstructured data by providing a simple to understand processing and communication method to the user, enabling a quick application of a "query" on the data set. Additionally, the design of the frameworks provide fault resiliency to the execution, enabling the use of commodity computing systems while scaling to very large data sets. All combined, these frameworks have made Big Data Analytics ubiquitous.

There is great value in the ability to access large data sets, but there is additional value in the ability to do so with extreme speed. Consider the impact to the Data Analyst if the response time of the big data query were in seconds or minutes, instead of hours or days. The effect would be to keep the Analyst engaged in the problem, prompting more and varied queries. The increase in performance also greatly enhances the ability of the Analyst to test new methods, leading to new insights into the data sets.

Consider then the value of near-real time or real-time analytic computations on live data. This performance level would enable a new concept of operations in Activity-Based Intelligence (ABI) and similar missions. For example, real-time data analytics could influence the collection of the very data being analyzed. In an alternate application, real-time data analysis could be used to influence the behavior of that which is the subject of data collection in the first place.

The combination of the productivity and the achievement of near-real-time or real-time performance defines High Performance Data Analytics (HPDA). The following sections address a solution to achieving HPDA. The first section describes an approach based on porting the open source analytic frameworks to HPC systems. This porting process, however, must re-define the fundamental design pattern of the open source frameworks in order to take advantage of the HPC performance capabilities. The next two sections provide the results of applying this process to the Apache Hama graph analytic open source framework, when ported to an HPC InfiniBand connected Cluster. First, the performance of the modified Hama is evaluated using a synthetic communication benchmark called CBench. Next, the framework as a whole is evaluated by executing a real-world Community Detection application against an 8 million vertice graph derived from Twitter data. By employing the proper porting techniques, a performance increase of greater than 10x is achieved. HPDA is not a stand-alone capability, but one that is integrated with other systems to fully impact missions. The final section below discusses a third goal of HPDA which is integration into the enterprise.

3. The Approach: Open Source Frameworks + HPC

The proliferation of the open source frameworks for big data analytics motivates their use as a baseline for HPDA. These are the frameworks that analysts are using as they experiment with new techniques on ever growing data sets. Users then leverage the natural growth in functional capability being constantly developed by the open source community. A natural path for HPDA is to improve the performance of the frameworks by hosting them on higher performing hardware, such as HPC systems.

Simply re-hosting an application on an HPC system doesn't guarantee improved performance. In fact, there is a fundamental difference between the design pattern used by many open source frameworks and the design pattern used by HPC applications, which precludes such performance gains. From the beginning, the philosophy used to guide the design of open source big data frameworks has been to use a distributed programming model. This model provides system fault resilience in software to enable the framework to scale to very large data sets on less reliable commodity hardware. This is the correct approach on commodity hardware.

In contrast, capability class HPC systems are designed for fault tolerance at the system level, and do not require the additional software layers found in the distributed model. A more appropriate programming model for HPC systems is the parallel programming model, which focuses on only the functions required by the application. In order to achieve the performance of the HPC system, the framework should be viewed as an application, and ported to the HPC system. The software layers added in the framework for fault resiliency are removed, providing higher performance. This approach maintains the user-facing application programming interface (API), making the transition to the HPC system seamless.

There are two key enablers in the design of existing open source analytic frameworks which support efficient hosting on an HPC system. First, the fault models that the software resilience layers target are primarily in the data interfaces between the parallel processing nodes. Some key interfaces include the data transfer and barrier synchronization between compute nodes, and data input/output to the nodes. These areas are precisely where HPC systems excel. The second enabler is that the implementation of the open source frameworks follows a very modular object-oriented approach. This approach encapsulates the framework functions into very well defined classes. This means that the software fault resilience layers are encapsulated within the function classes where faults are expected to occur, namely in the data movement interfaces. Surgical replacements of these classes with implementations that use the HPC parallel design pattern remove the software overheads and achieve the HPC performance levels. The remainder of the framework, especially the users' API, remains intact. The following section describes the results of applying this approach in porting the open source Apache Hama graph analytic framework to an HPC InfiniBand Cluster.

4. Proof of Concept: The Apache Hama Test Case

Graphs provide a natural representation of unstructured but related data. However, many characteristics of real-world graphs make them difficult to process in parallel. In particular, Graphs constructed from many real-world data sets follow a power-law distribution in node-degree, resulting in low diameter and non-partitionable graphs. Furthermore, the algorithms that are of most interest for graph analytics typically have low vertex computation to edge communication ratios, further stressing the fault-prone communication infrastructure of the commodity computing systems. In contrast, HPC systems typically provide high bandwidth communication and fast synchronization mechanisms. These characteristics make graph analytics an ideal test case for porting representative frameworks to an HPC system.

There are several examples of proprietary and open source graph analytic frameworks, such as Googles' Pregel, Apache Giraph, Graph Lab, and Graph Processing System. For this exercise, the open source Apache Hama [1] graph analytic framework was chosen as a test case, and ported to an HPC InfiniBand interconnected Cluster. Apache Hama is built on Apache Hadoop (Core, HDFS, ZooKeeper), and provides a representative baseline for many open source frameworks.

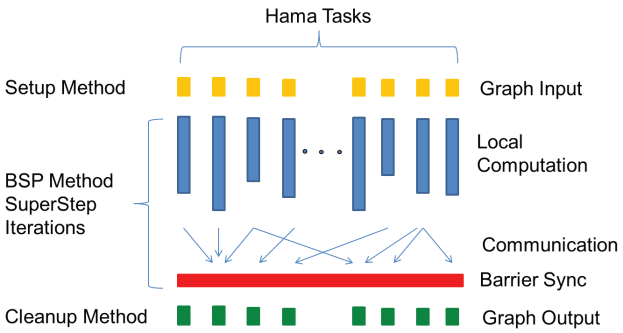

Figure 1. Hama Bulk Synchronous Processing (BSP) Model

Feature	Description
NUMBER OF NODES	128
NODE CONFIGURATION	Dual Socket SMP
PROCESSOR	x86_64 26xx class, 8-core, dual threaded
MEMORY	64 GB/node
INFINIBAND	Quad FDR
NODE OPERATING SYSTEM	Linux
SYSTEM RESOURCE MANAGER	SLURM

Table 1. Test HPC InfiniBand Cluster Configuration

5. Apache Hama Overview

Apache Hama provides a parallel processing framework modeled after the Bulk Synchronous Processing (BSP) model, as shown in Figure 1. The primary processing is performed iteratively as a sequence of SuperSteps, consisting of local computation (Compute), communication (Communication), and a barrier synchronization (Barrier Sync). Hama also supports a graph analytic specific "vertex centric" API, with its Graph Package. In this case, the local computation is specified for a single vertex, the communication is specified from a vertex to neighboring vertices connected by edges in the graph, the barrier is between all vertices in the graph. Hama also provides for initial graph input during a prefix Setup method, and a similar algorithm results output during a postfix Cleanup method.

Apache Hama makes use of the Apache ZooKeeper project to perform basic bookkeeping on the parallel tasks, as well as to implement the Barrier Sync function. For this exercise, Apache Hama version 0.6.4, was combined with Cloudera's Hadoop distribution, CDH 4.5.0 [2] and stood up on an InfiniBand Cluster, as described in Figure 1.

6. The HPC InfiniBand Cluster

A large percentage of capability class HPC systems use InfiniBand technology to interconnect high performance compute nodes. The target system for this exercise is described in Table 1.

7. Initial Evaluation

The Apache Hama framework is written in Java, as are the BSP and Graph Package user applications. There is a very minimal framework API for sending and receiving messages, and performing the barrier synchronization of the SuperStep. The data input and output follows the Hadoop model. Several classic graph analytic kernels were implemented. These kernels include Breadth First Search, Connected Components, PageRank, Clustering Coefficients, Graph Pattern Matching (based on Strict Simulation), Single Source Shortest Path, All Source Shortest Path, and Betweenness Centrality (vertex and edge centrality). The implementations of these test kernels spanned all three programming models of Hama (BSP, Graph Package, and C++ Pipes). Two of the test kernels, CBench and BMLPA are described below. The expressive semantics of Java resulted in very short development times and compact code for individual graph kernels. In short, the Productivity of the framework was very good.

For performance analysis of Apache Hama, several suites of tests were generated for the various graph analytic kernels, primarily varying data set sizes and degree of parallelization. Across the board, it was noted that the Barrier Sync and the Communication component times dominated the total runtime of each kernel. The balance between the two components varied depending on the kernel, but in all cases the total runtime was dominated by these basic framework services. An analysis of the data showed that the performance of these two components was very poor (orders of magnitude) compared to analogous code using the native MPI libraries on the InfiniBand Cluster. This data is provided in Figure 4 and Figure 5.

8. A Better Hama

The decision was made to modify the Apache Hama framework to improve the performance of the Communication and Barrier Sync to leverage the strengths of the HPC system. Analysis of the Hama framework showed that the Barrier Sync and Communication functions are implemented as very modular Java class objects. Furthermore, the software layers required for fault resilience in a distributed environment were easily identified.

Figure 2 depicts the original Apache Hama block diagram for the Barrier Sync and Communication (messaging) functions. Hama uses ZooKeeper to implement the Barrier Sync function. Several ZooKeeper servers are started in the system, and each Hama task must connect to one of the servers to join the barrier. The ZooKeepers then communicate to complete the barrier, and notify the Hama tasks. For fault resilience, the ZooKeeper servers all must keep a consistent state of all Hama tasks that have entered the barrier. As Hama tasks join the barrier, the ZooKeeper servers begin the process of replicating their state among each other, adding to the overhead. In the event that a ZooKeeper server is lost (due to network failures, server failures, etc.) the remaining ZooKeeper servers can continue serving the Hama tasks. While this is the correct philosophy for a commodity environment, it is not necessary in an HPC environment.

Figure 3 depicts the modified Apache Hama block diagram for the Barrier Sync and Communication (Message) functions. A simple Barrier Sync function was implemented as follows. Instead of launching a small number of ZooKeeper servers, an MPI-based server was developed, and launched one per compute node in the InfiniBand Cluster. To join a Barrier Sync, each Hama task contacts the local MPI server via a Barrier pthread. The MPI servers wait until all local Hama tasks have joined, then call a global MPI barrier function. When the global MPI barrier returns, the MPI servers respond back to the Hama tasks. The modifications to the Hama framework were minor, and consisted of modifications to the Sync class which formerly interfaced with ZooKeeper to now interface with the MPI server. The only other modification was to the initialization code which would normally bring up the ZooKeeper servers for the Barrier Sync function. This code was changed to start up the MPI server.

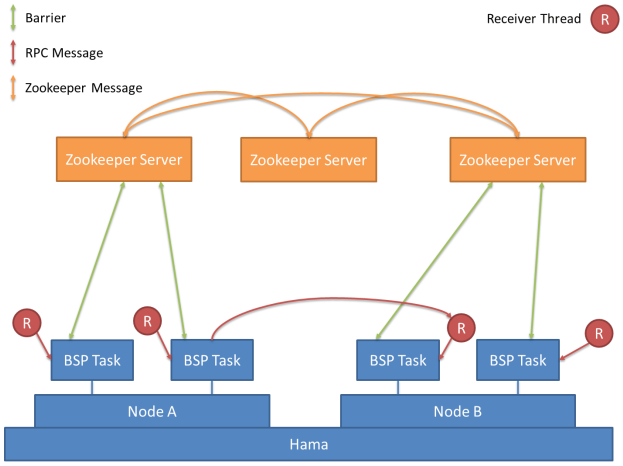

Figure 2. Baseline Apache Hama Barrier Sync and Communication Functions

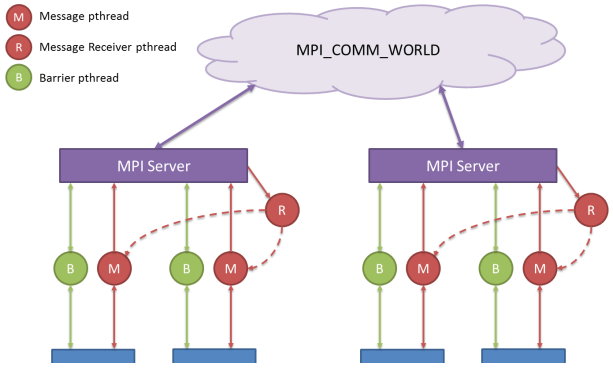

Figure 3. Improved Apache Hama Barrier Sync and Communication Functions

The replacement of the communication function was similarly focused on just the part of the interface moving the data between Hama tasks. As shown in Figure 2, the baseline Hama uses a Remote Procedure Call (RPC) for this data movement. Each Hama task would contact all of the other Hama tasks directly for which it had data to transfer. A Receiver thread was used to capture incoming messages for later use by the destination Hama task. This RPC approach includes several exchanges of control, status, and other "chatter" to implement a reliable communication protocol in a commodity environment. Similar to the Barrier Sync implementation in MPI, the communication functions were modified to connect to the local MPI server as shown in Figure 3. A Hama task would send all of its outgoing messages to the local MPI Server. The local MPI Server would then aggregate messages to other destination compute nodes, and then transfer the appropriate data directly to the MPI server on each destination node. On the destination side, the MPI Server used a Receiver pthread to capture all incoming messages for local Hama tasks. Finally, each Hama task would use its own Message pthread to pull down its messages from the MPI Server Receiver pthread. The use of pthreads for the modified Barrier Sync and Communication functions enabled the use of hardware shared memory support on the HPC compute node, further improving the performance.

This MPI-based approach, while not an ideal solution, was selected to support a quick evaluation of alternate approaches to see if any further performance bottlenecks are hidden behind these two functions. As the performance benefits are quantified, the goal would be to develop similar HPC-related solutions that are robust and maintainable, and release those implementations to the open source community. If properly designed, these implementations may be substituted into the standard distribution via configuration settings at runtime.

9. Performance Characterization using a Synthetic Benchmark

To accurately test the MPI-based improvements, a synthetic benchmark, the Communications Benchmark or CBench was developed. The key CBench parameters are given in Table 2.

In order to evaluate the performance of the modified Barrier Sync function, a parameter sweep scaling test using CBench was configured and executed against both versions of the Hama framework running on the InfiniBand Cluster. CBench was configured as (T=??, P=T, N=1, M=1024, I=200). Each compute node supported 16 Hama tasks. During each SuperStep of the test, each CBench task generated a 1KB message to every other task in the configuration. The results are depicted in Figure 4. As the number of Hama tasks (T) grows, the time Barrier Sync time also grows. In the best case, this growth would be logarithmic in the number of tasks. For the ZooKeeper Sync, the growth is slightly more than linear. The MPI Sync scales much better, achieving a performance Speedup of more than 4 for small numbers of tasks, and stabilizing from 2-3 for the range of task configurations tested. Extrapolating to larger numbers of tasks, it is expected that the MPI Sync would continue to increase its performance gains over the Zookeeper Sync implementation.

A similar approach was used to evaluate the performance of the modified Communication function. A CBench parameter sweep test was configured as (T=512, P=T, N=1, M=??, I=200), and executed on both versions of the Hama framework running on the Infiniband Clusters. During each SuperStep of the test, each CBench task generated a message of the current size and sent it to every other task in the configuration. The results are depicted in Figure 5. For small messages, less than about 8KB, both versions of Hama performed well, with the MPI Communication version achieving a speedup of about 5. However, when the message size grows to 8KB and beyond, the baseline Hama RPC Communication performance drops considerably while the performance of the MPI Communication version scales with the message size. The 8KB message size is related to the underlying Message Transfer Unit (MTU) of the Infiniband. The MPI Communication version achieves speedups of 10-25 for larger messages. The non-uniform performance spikes in the baseline Hama are also related to the MTU size. Changing the MTU size changes where the spikes occur, but the overall performance profile is similar.

10. Performance Achievements with Community Detection in a Twitter

In order to fully test the performance benefits of the improved Hama, a real-world application was built in the framework and tested against real data. The application chosen for this effort was a Community Detection kernel based on label propagation named Balanced Multi-Label Propagation Algorithm (BMLPA) [3] taken from the open literature. The algorithm reads in a graph with vertices (V) and edges (E). The vertices are spread across Hama tasks (T). Initially, each vertex begins with a set of labels provided in the input graph. In each iteration, a vertex sends its current list of labels to all neighbors connected by edges. In turn, each vertex receives label lists from its neighbors on its incoming edges. The lists are sorted and ranked by frequency of occurrence. The vertex keeps the labels which occur above some cutoff frequency termed the “Belonging Probability (b).” The algorithm terminates when the rate of label updates drops below an intrinsic constant.

As a test data set, a graph was generated from Twitter data. The each vertice of the graph was a unique Twitter user. An edge from user i to user j was added if user i generated a tweet that “mentioned” user j, either through a directed @ tweet, retweet, etc. If the edge-generating tweet included a hashtag, that hashtag was added to initial label list for user i. For this test, a graph with ~8 million vertices was extracted from Twitter data (April 2014). The resultant graph had

Parameter	Description
NUM_TASKS, T	Number of BSP Tasks executing
NUM_PEERS, P	Number of Peers with which a Task will randomly select to communicate
NUM_MSGS, N	Number of messages to send between communicating Peers
MSG_SIZE, M	Message Size in bytes
NUM_ITERS, I	Number of BSP SuperSteps to execute

Table 2. CBench Parameter Descriptions

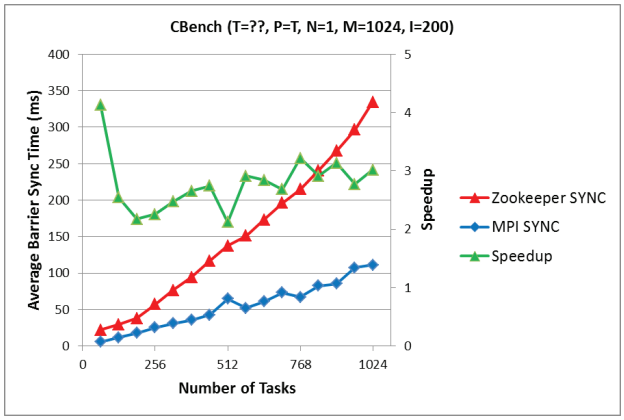

Figure 4. MPI Barrier Sync Performance Speedup and Scalability

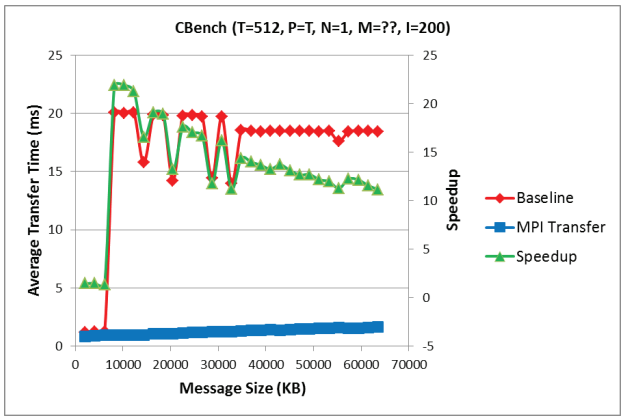

Figure 5. MPI Communication Performance Speedup and Scalability

~10 million edges. Anecdotally, there were many small user communities, often in hub or chain configurations. There were a few vertices that were mentioned a large number of times, thus creating large communities.

The BMLPA test cases were parameterized as (T=256|384|512, b=0.05|0.10|0.20|0.40) and executed on 64 compute nodes, each node executing 8 Hama tasks. Figure 6 depicts the performance improvements in the BMLPA Community Detection application combined with the Twitter data. For a Hama configuration of 256 tasks, the speedup was about a factor of 6, with 384 tasks the speedup was about a factor of 10, and at 512 tasks the speedup varied with the Belonging Probability b. Across all tested configurations, the MPI Hama out performed the original Hama.

The variation in performance from run to run is directly linked to the average amount of data transferred from one Hama task to another during the execution. Figure 7 provides the average message bundle transferred task-to-task during each test case. Correlating the data in Figure 7 to the data in Figure 5 explains the application speedups reported in Figure 6. For example, when T=512 and b=0.40, the average message bundle size is about 4KB. This message size shows that the peak speedup in the transfer component only of the application is capped at 5. Ahmdahls law dictates that the total speedup will be less than this peak due to other execution components being the same between the original and the modified Hama versions. As a second example, keeping T=512 but for b=0.05, the average message size is about 8KB, which shows a peak speedup in the transfer component of over 23. Hence the total speedup is greater. This also indicates that the message transfer time is a dominant component of this application. Achieving a significant speedup when the message size is small (T=512, b=0.10) also indicates that the Barrier Sync time is a significant contributor as well.

In comparing the performance of the algorithm runs using the ZooKeeper Sync and RPC Communication versus the MPI Sync and Communication versions of Hama, the MPI version clearly is a win, reducing the total BSP run time on

a real application by a factor of 10. This performance gain underscores the importance of efficient Barrier Sync and Communication implementations in the Hama (or any) framework.

11. Work in Progress

In order to fully evaluate the MPI server approach, the BMLPA Community Detection application will be chained with two other applications. First, the larger communities identified by the BMLPA algorithm will be individually processed by a Betweenness Centrality algorithm to identify "leaders" for each community. Next, the Twitter graph annotated with community and community leader data will be processed by a Force Directed Layout [4] algorithm to produce 2-D and 3-D displays of the annotated community graph. All algorithms are implemented in Hama with BSP or Graph Package programming models. Full performance analysis of the algorithms will be used to further evaluate the success of the MPI-server approach.

12. Next Steps: Integrating HPDA into the Enterprise

Achieving real-time performance with open source data analytic frameworks is the first step towards the end goal of enhancing the enterprise mission. The real value comes with the integration of the analytic computations with "the mission" to enhance the value of the mission. There are three key thrust areas that must be addressed.

The first area is to continue the work in the open source community for achieving interoperability of multiple analytic frameworks in the enterprise. Within a given mission pipeline, each step may require a different type of analysis, hence a different framework to be employed. Data and results are passed from one step to the next, from one enterprise resource to the next. As new frameworks are ported to HPC systems, the modifications should be fed back to the open source community.

Next, the data analytics must share data with legacy systems in the enterprise. There are several aspects to this problem, including the ability to quickly stream data between legacy file systems and the standard HDFS big data file systems, as well as to directly access the legacy file systems. HPC file systems provide sufficient parallelism and performance to support this approach. Smart data movement technologies also play a role.

Finally, the data analytics frameworks and applications must be able to interact directly with existing HPC resource management and workflow tools. HPC resources are expensive, and must be shared with other operations. The goal is to dynamically deploy a data analytic framework to an HPC system as needed, run the analytic applications, then tear down the framework and release the HPC resource back to the enterprise. This capability supports the dynamic allocation of HPC resources, improving the value to the enterprise. The final vision is to provide the ability to use the enterprise workflow environment to dynamically stand up a mission pipeline, including contributions from open source data analytic framework running in real-time.

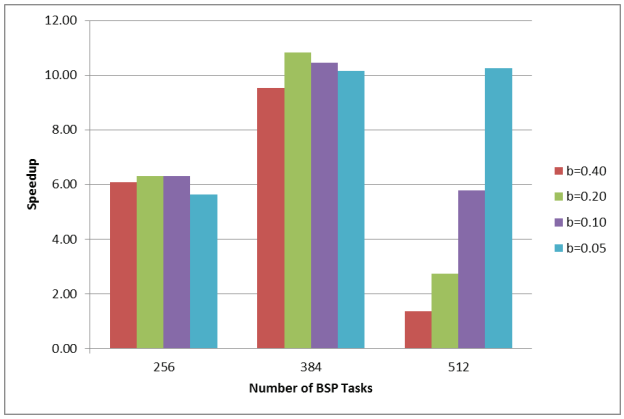

Figure 6. Achieved Speedup with the Community Detection applied to Twitter Data

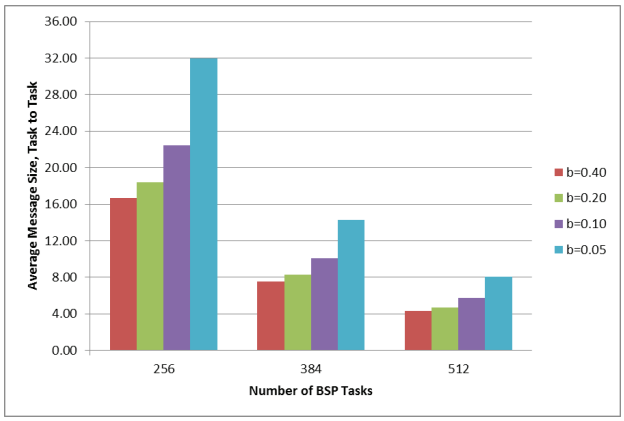

Figure 7. Average Task-to-Task Message Sizes in the Community Detection Application

13. General Dynamics Mission Systems Background

General Dynamics Mission Systems develops resilient mission systems and products for high value information and cyber platforms. With unsurpassed mission knowledge, an open approach and unrivaled portfolio, we advance cybersecurity for our nation's defense, intelligence and infrastructure. As technology evolves, everything is more connected and information is increasingly valuable and vulnerable. To outpace and outsmart pervasive and sophisticated threats, General Dynamics is leading a revolution in the approach to cybersecurity.

For more than 35 years, we have been solving our customer's most challenging problems through the design, development, and deployment of HPC systems.

[1] "The Apache Hama Project," 2014. [Online]. Available: http://hama.apache.org.

[2] "CDH," 2014. [Online]. Available: http://www.cloudera.com.

[3] Y.-F. L. S. G. H.-Y. W. S.-F. T. Zhi-Hao Wu, "Balanced Multi-Label Propagation for Overlapping Community Detection in Social Networks," Journal of Computer Science and Technology, vol. 27, no. 3, pp. 468-479, 2012.

[4] E. M. R. Thomas M. J. Fruchterman, "Graph Drawing by Force-Directed Placement," Software-Practice & Experience, vol. 21, no. 11, pp. 1129-1164, 1991.

[5] S. F. Andrea Lancichinetti, "Benchmarks for Testing Community Detection Algorithms on Directed and Weighted Graphs with Overlapping Communities," Physics Review, 2009.

[6] "Twitter," [Online]. Available: https://twitter.com.

GENERAL DYNAMICS
Mission Systems

12450 Fair Lakes Circle
Fairfax, VA 22033

www.gdmissionsystems.com

Cleared for external release #1905, October 2014