To 4,000 Compute Nodes and Beyond: Network-aware Vertex Placement in Large-scale Graph Processing Systems

Karim Awara
KAUST
Thuwal, Saudi Arabia

Hani Jamjoom
IBM Watson Research Center
Yorktown Heights, NY

Panos Kalnis
KAUST
Thuwal, Saudi Arabia

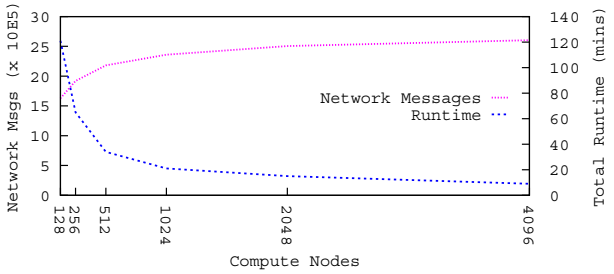

Figure 1: Change in messages sent across compute nodes and total runtime.

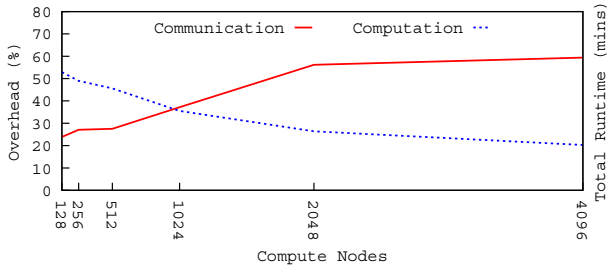

Figure 2: As BSP system scales, the execution per compute node decreases, but the communication increases. Compute time excludes initialization cost and overhead of combiners.

Categories and Subject Descriptors

H.2.8 [**Information Systems**]: Data mining; D.4.8 [**Performance**]: Measurements

Keywords

Graph Mining Systems; Bulk Synchronous Parallel; Vertex Placement; Network Topology; Extreme Scaling

1. INTRODUCTION

The explosive growth of "big data" is giving rise to a new breed of graph processing systems, such as Pregel [5]. Pregel and its derivatives follow the Bulk Synchronous Parallel (BSP) programming model, where vertices send messages asynchronously to other vertices over a series of globally synchronized supersteps (iterations). As these systems scale across large infrastructures, communication overhead dominates the cost of the applications' runtime.

This poster describes our work in characterizing and minimizing the communication cost of BSP-based graph processing systems when scaling to 4,096 compute nodes. To the best of our knowledge, this is the largest deployment to date. Existing implementations generally assume a fixed communication cost. Large graphs are first partitioned into smaller subgraphs (clusters), which are then mapped onto the compute infrastructure, irrespective of the network cost between compute nodes. This is sufficient in small deployments as the BSP programming model masks small variations in the underlying network by overlapping computation and communication. Because the per-vertex computation is typically small, in large scale deployments, variations in network topologies and costs cannot be fully masked.

There are numerous systems that tackle the scalability problem. Most notably is PowerGraph [1], which replicates highly connected vertices to reduce cross partition communication. As the system scales to thousands of compute nodes, random replication does not fully address network cost problem. Also recently, Hoefler and Snir [2] evaluated different topology mapping strategies. While promising, their approach in using the Reverse Cuthill-McKee (RCM) algorithm cannot scale to map millions of vertices.

In this poster, we first quantify the impact of network communication on the total compute time of a BSP system. We then propose an optimization that improves the scalability of RCM; we describe an improved vertices replication strategy to further reduce the communication overhead.

2. COMMUNICATION IMPACT

We ran our experiments on an IBM Blue Gene/P supercomputer, using 4 racks, each with 1024 PowerPC-450 CPUs (4 cores at 850MHz and 4GB RAM per CPU). While individual compute nodes are relatively slow in today's standards, the focus of the work was on studying the *relative* impact of communication on the overall runtime. More importantly,

SIGCOMM'13, August 12–16, 2013, Hong Kong, China.
ACM 978-1-4503-2056-6/13/08.

the Blue Gene/P allowed us to conduct experiments on a broad range of compute nodes, which would otherwise be more difficult to perform on an x86 compute cloud. We used Mizan [3], a BSP system that implements the Pregel programming model, to run the PageRank algorithm on the `arabic-2005` dataset [4], with approximately 23M vertices and 640M edges. We varied the number of compute nodes from 256 to 4096. In each experiment, the dataset is partitioned using a hash-based scheme and distributed across the participating compute nodes. Finally, we instrumented Mizan to measure the computation and communication time

Figure 1 shows the increase in messages as the number of compute nodes is increased. Similarly, Figure 2 shows the percentage of time spent in performing computation and communication. The result (right y-axis in Figure 1) is a gradual plateau of improvements in end-to-end runtime. These results confirm our intuition; we are in the process of testing much larger datasets and other graph mining algorithms. Our goal is to limit the effects of communication overhead in large scale deployments.

3. NETWORK-AWARE VERTEX PLACEMENT

Basic Approach. It is well known that matrices can represent both graphs and communication costs between compute nodes. In our setting, we have two matrices: the first, π_G^A, captures the expected communication between any two vertices in input graph G when running algorithm A; the second, π_C, captures the communication costs between two compute nodes.

These two adjacency matrices are sparse. Applying the Reverse Cuthill-McKee (RCM) algorithm reduces the bandwidth of these sparse matrices [2]. The result is two matrices that have mostly zero values away from the diagonal. For π_G^A, it implies that vertices that communicate with each other are placed closer to each other. Similarly, for π_C, it implies that compute nodes with higher bandwidth (or lower latency) are placed closer to each other. The last step then is to simply partition the matrix and map π_G^A onto π_C.

Practical Issues in using RCM. In this poster, we highlight three practical issues. First, π_G^A is potentially very large. For a billion node graph, π_G^A becomes billion by billion in size. Second, each element in the matrix represents the expected communication (number of messages) between the vertices when running an arbitrary algorithm A. Third, in cases where the computation is running in the cloud, π_C is not known a priori and needs to be measured.

System Implementation of Placement Algorithm. Figure 3 highlights our approach in solving the first challenge. We assume that the remaining two challenges, while important, can be approximated. We first (Step 1) identify the edge degrees for all vertices and subsample the input graph to include only high-degree vertices. Then, we run RCM topology mapping on the smaller graph (Steps 2 and 3). The partitioned graph may still have high communication around the edges. We use vertex replication (Step 4) to minimize communication across partitions. Finally, we bring back low degree vertices (Step 5).

Our use of RCM has two key advantages over traditional partitioning techniques (like min-cut). First, it provides a natural ordering of partitions, such that partitions that communicate with each other will be placed closer to each other. Second, for the same reason, it identifies good candidates for replication. We are currently in the process of evaluating our algorithm on large datasets.

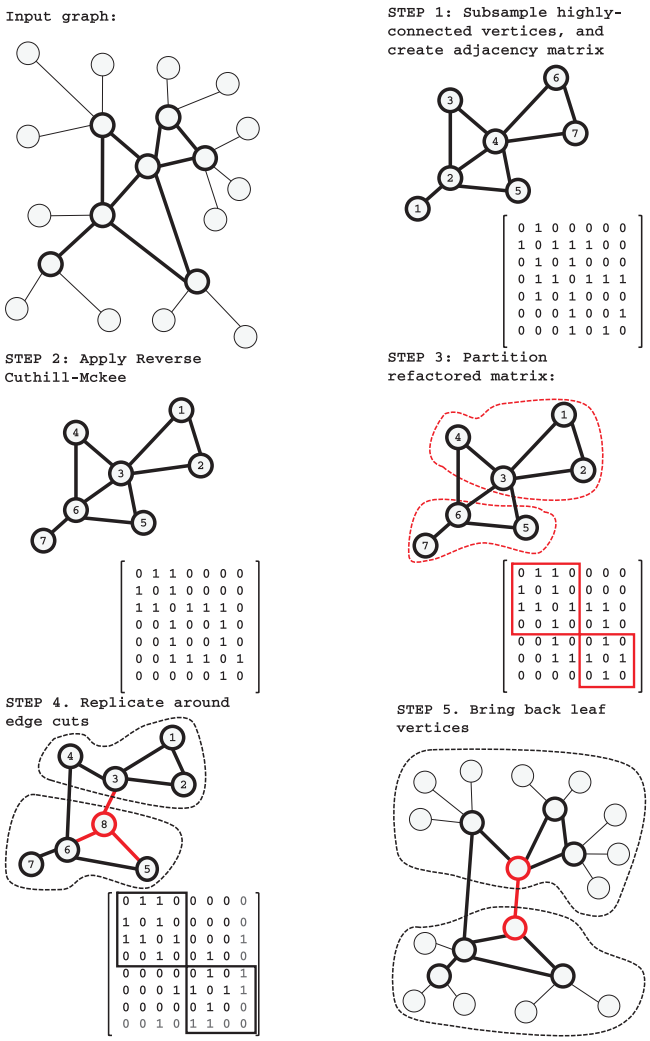

Figure 3: Example algorithm

4. REFERENCES

[1] Gonzalez, J. E., Low, Y., Gu, H., Bickson, D., and Guestrin, C. PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs. In *Proceedings of USENIX OSDI* (Hollywood, Oct 2012).

[2] Hoefler, T., and Snir, M. Generic Topology Mapping Strategies for Large-scale Parallel Architectures. In *ACM ICS* (Tucson, AZ, June 2011).

[3] Khayyat, Z., Awara, K., Alonazi, A., Jamjoom, H., Williams, D., and Kalnis, P. Mizan: A System for Dynamic Load Balancing in Large-scale Graph Processing. In *ACM EuroSys* (Prague, April 2013).

[4] The Laboratory for Web Algorithmics. `http://law.di.unimi.it/index.php`, 2012.

[5] Malewicz, G., Austern, M. H., Bik, A. J., Dehnert, J. C., Horn, I., Leiser, N., and Czajkowski, G. Pregel: A System for Large-scale Graph Processing. In *Proceedings of ACM SIGMOD* (Indianapolis, IN, 2010).