All-Pairs Shortest Paths in Spark

Charles Y. Zheng and Jinshu Wang
Department of Statistics
Stanford University
Stanford, CA 94305
{snarles, jinshuw}@stanford.edu

Arzav Jain
Department of Computer Science
Stanford University
Stanford, CA 94305
arzavj@stanford.edu

Abstract

We propose an algorithm for the All-Pairs-Shortest-Paths (APSP) problem suitable for implementation in Spark, and analyze its performance. We begin by considering distributed Floyd-Warshall, as proposed by Kumar and Singh (1991). Distributed Floyd-Warshall has asymptotically optimal scaling and can be implemented in Spark by using BlockMatrix to represent the APSP distance matrix. However, we observe that its implementation in Spark suffers from poor performance for medium-sized problems due the large number of global updates of the APSP distance matrix required for the algorithm. Since the lineage of the algorithm grows with the number of vertices n, it becomes necessary to use a proportional number of checkpoints which further impacts the efficiency of the algorithm. This motivates the consideration of an algorithm for APSP which requires fewer global update steps. We adapt an approach by Solomonik et al. (2013) based on the "divide and conquer" algorithm for APSP. Our algorithm reduces the number of global updates by a factor of b, where the block size b determines the amount of computation done in each iteration. By adjusting the block size b we obtain a favorable tradeoff between checkpointing costs and computation cost per iteration, resulting in far improved performance compared to Distributed Floyd-Warshall.

1 Summary

For the convenience of the grader we present an overview of our approach and our results. The rest of the paper gives a detailed explanation of the results in this section.

1.1 Problem Specification

Let $G = (V, E)$ be a graph with n vertices. Assume the input is given in the form of the adjacency matrix A of the graph stored as a `BlockMatrix` with equally sized square blocks. Specifically define the adjacency matrix A as a square matrix with dimension $n = |V|$, and entries

$$A_{ij} = \begin{cases} w_{i,j} & \text{if } (i \to j) \in E \\ 0 & \text{if } i = j \\ \infty & \text{if } (i \to j) \notin E \end{cases}$$

Let b be the size of the block, and let $n = b\ell$ so that ℓ^2 is the number of blocks. Write

$$A = \begin{pmatrix} A^{11} & A^{12} & \cdots & A^{1\ell} \\ A^{21} & A^{22} & \cdots & A^{2\ell} \\ \vdots & \vdots & \ddots & \ddots \\ A^{\ell 1} & A^{\ell 2} & \cdots & A^{\ell\ell} \end{pmatrix}$$

so that A^{ij} is the (i,j)th block in the `BlockMatrix`.

The output is given by the APSP distance matrix S, where

$$S_{ij} = \begin{cases} \text{weight of shortest path} & \text{if there exists a path } i \to j \\ 0 & \text{if } i = j \\ \infty & \text{if there is no path } i \to j \end{cases}$$

Let S be stored as a `BlockMatrix` with the same dimensions and block sizes as A, so that

$$S = \begin{pmatrix} S^{11} & S^{12} & \dots & S^{1\ell} \\ S^{21} & S^{22} & \dots & S^{2\ell} \\ \vdots & \vdots & \ddots & \vdots \\ S^{\ell 1} & S^{\ell 2} & \dots & S^{\ell\ell} \end{pmatrix}$$

Let p be the number of workers. Let M be the memory of ech worker. Suppose each worker holds K contiguous blocks. It must be the case that $K < M/b^2$. In fact, K is even smaller because each worker will have to hold additional data in memory.

1.2 Scaling

We consider the scaling $n \to \infty$ and $p \to \infty$. We do *not* assume a sparse graph G, so the number of edges can scale as $E \sim n^2$. However b must be constant since we assume each block must fit in memory.

1.3 Notation

Given an $n \times k$ matrix A and a $k \times m$ matrix B, define the *min-plus* product $C = A \otimes B$ by

$$C_{i,j} = \min_{l=1}^{k} A_{il} + B_{lj}$$

for $i = 1, \dots, n$ and $j = 1, \dots, m$.

Define APSP(A) as the all-pairs-shortest-distance matrix for adjacency matrix A. For example, APSP(A) is obtained by running the Floyd-Warshall algorithm on A.

1.4 Algorithm

The algorithm consists of an *outer loop* with $\ell = n/b$ iterations. Each iteration culminates in the global update of the `BlockMatrix` S containing the intermediate values of the APSP distance matrix. Each outer loop iteration involves the execution of three distributed subroutines in sequence, called the A-step, the B-step and C-step. In addition, after every q iterations, the `BlockMatrix` S is checkpointed.

We first give an shorthand description of the algorithm without explicitly specifying the Spark operations used in each step or what data needs to be communicated at each step. In the analysis, we expand each step to describe the specific Spark operations needed, including the broadcasts, joins, etc. needed to transfer the necessary data across workers. Do note that $S^{(0)}, S^{(1)}, \dots, S^{(\ell)}$ refer to the sequence of `BlockMatrix` objects storing the results of each iteration.

Algorithm 1 Distributed Block APSP (shorthand)

function BLOCKAPSP(Adjacency matrix A given as a `BlockMatrix` with ℓ row blocks and ℓ column blocks)
  $S^{(0)} \leftarrow A$
  for $k = 1, \ldots, \ell$ **do**
    [A-step]
    $S^{kk(k)} \leftarrow \text{APSP}(S^{kk(k-1)})$
    [B-step]
    for $i = 1, \ldots, \ell,\ j = 1, \ldots, \ell$ **do** *in parallel*
      if $i = k$ and $j \neq k$ **then**
        $S^{kj(k)} \leftarrow S^{kk(k)} \otimes S^{kj(k-1)}$
      end if
      if $i \neq k$ and $j = k$ **then**
        $S^{ik(k)} \leftarrow S^{ik(k)} \otimes S^{kk(k)}$
      end if
    end for
    [C-step]
    for $i = 1, \ldots, \ell,\ j = 1, \ldots, \ell$ **do** *in parallel*
      if $i \neq k$ and $j \neq k$ **then**
        $S^{ij(k)} \leftarrow S^{ik(k)} \otimes S^{kj(k)}$
      end if
    end for
    [D-step]
    if $k \equiv 0 \mod q$ **then**
      Checkpoint $S^{(k)}$
    end if
  end for
  Return $S = S^{(n/b)}$, the APSP matrix in `BlockMatrix` form
end function

1.5 Optimality

The single-core cost of Floyd-Warshall, the best known single-core algorithm for APSP, is $O(n^3)$. A perfectly distributed form of Floyd-Warshall therefore has a total runtime of $O(n^3/p)$, in the asymptotic regime $n \to \infty$ and $p = O(n)$. Our algorithm achieves the same asymptotic runtime of

$$O\left(\frac{n^3}{p} + \frac{n}{b} + \right)$$

for details see section 3.

One can also consider the *communication cost* scaling in terms of the amount of data tranferred over the network. The paper by Solomonik et al. (2013) derived a theoretical lower bound on the communication cost of APSP as $\Omega(\frac{n^2}{p^{2/3}})$ words. In comparison, our algorithm communicates a total of $O(n^2\sqrt{p} + \frac{n}{b}\sqrt{p})$ words, which is worse by a power of p.

1.6 Communication Cost and Type

We analyze the *bandwidth* (total words sent) and the type of communication in each step of the algorithm. The A-step involves a one-to-one communication of a matrix of size $b \times b$ from a worker to the driver, involving a bandwidth of $O(b^2)$ words. The B-step involves a one-to-all broadcast of a matrix of size $b \times b$ from the driver to $\sqrt{p}$ workers, hence a bandwidth of $O(b^2\sqrt{p})$ words. The C-step involves an all-to-all communication (a map-side join) where each worker recieves two $n/\sqrt{p} \times b$ matrices, and therefore entails a bandwidth of $O(nb\sqrt{p})$. Therefore the per-iteration bandwidth is $O((1 + \sqrt{p})b^2 + \frac{nb}{\sqrt{p}})$. The total bandwidth for the algorithm is $O(n^2\sqrt{p}b + n(1 + \sqrt{p})b)$. See section 3 for the derivation of the bandwidth per step.

2 Background

3 Analysis

In each step we give the *computational cost* (computation done on each worker), the *bandwidth* (number of words sent) and the total *runtime* of the entire step. We give an exact (non-asymptotic) analysis given the following assumptions:

1. The time it takes to run Floyd-Warshall on a local matrix of size $b \times b$ is given by $\kappa_F b^3$
2. The time it takes to locally perform min-plus multiplication on matrices of size $a \times b$ and $b \times c$ be given by $\kappa_M a b^2 c$
3. Separate the cost of communication and computation, so that sending messages and receiving messages is not included in the computational cost.
4. Time taken to send one message consisting of m words from one machine (whether a worker or driver) to another machine is

   $$T(m) = \kappa_L + \kappa_T m$$

   where κ_L is a latency constant describing the time it takes for data to travel through the network and κ_T is the transmission time per word.
5. Each machine has a probability ϵ of instantaneously failing. The machine is instantly replaced, but all data in memory and disk is lost.
6. Let the time it takes to write to disk be given by $r\kappa_C m$, where κ_C is a constant for the disk write time and m is the amount of data each worker has to write. This is assuming the data is already on disk.
7. Assume no time cost for deleting data from disk.
8. We assume that the initial data A is stored on fault-tolerant backup, otherwise there is no guarantee that the job can complete. We assume a cost of $\kappa_B n^2$ to restore the cluster to the initial state from backup.

Note in particular that assumption 4 assumes a *uniform* network topology so the transmission rate from any worker to any other worker is equal. This is in contrast to the *grid* or *hypercube* topologies considered in most of the existing literature on distributed APSP.

Assumptions 5-8 deal with fault-tolerance. In the following we will start by giving an analysis of *fault-free iterations*, and then as we analyze the global cost, we incorporate the extra cost from faulty iterations.

3.1 A-step

3.2 B-step

3.3 C-step