Data Transfer for Distributed Analytics with Latency Constraints

Stephen Pasteris*, Shiqiang Wang†
*Department of Computer Science, University College London, UK, Email: s.pasteris@cs.ucl.ac.uk
†IBM T. J. Watson Research Center, Yorktown Heights, NY, USA, Email: wangshiq@us.ibm.com

***Abstract*—Tactical coalition applications are usually time-critical. Hence, it is important to consider the latency of communication pipelines in addition to throughput. In this paper, we study the problem of sending a given amount of data from one node to another node in the fastest possible time on a directed network, where edges have given bandwidths and transmission latencies. The two extremes of this problem are shortest-path routing and max-flow routing. Namely, when the amount of data is small, the bottleneck is latency rather than bandwidth so the problem is solved by shortest-path routing; and when the amount of data is large, the bottleneck is bandwidth so the problem is solved by max-flow routing. In general, the optimal routing strategy changes over time, starting with a max-flow strategy when a large amount of data remains to be transferred and then progressively turning into shortest-path when there is less data remaining to be transferred. We propose an efficient algorithm for solving this dynamic data transmission and routing problem. The methods developed in this paper is fundamental to incorporating the effect of latency in other more complex coalition problems.**

I. Introduction

We have a directed graph $G = (V, E)$. The vertices of the graph represent routers and the edges of the graph represent (direct) communication pipelines between the routers. For every directed edge $(v, w) \in E$ we have an associated "bandwidth", $\beta(v, w) \in \mathbb{R}^+$, and "latency", $\lambda(v, w) \in \mathbb{R}^+$. The bandwidth of an edge (v, w) is the number of bits per second that can be passed through the (direct) communication pipeline from v to w and the latency of an edge (v, w) is the time (in seconds) taken to send a bit from v to w via the (direct) communication pipeline from v to w.

The problem is as follows: we have Δ bits of data stored at a "source" vertex, $\perp \in V$, which need to be transferred to a "sink" vertex, $\top \in V$. We need to compute how to route the data so that this data-transfer takes the minimum possible time.

We assume, in this paper, that Δ is very large (infinite limit) which allows us to relax the problem so that a number of bits is now a continuous variable rather than discrete.

Without loss of generality we remove all edges of the form $(v, \perp)$ and $(\top, v)$ from E.

In order for our scheduling computation algorithm to be efficient we assume that all bandwidths on edges are elements of $\{\alpha k : k \leq \kappa\}$ for some $\kappa \in \mathbb{N}$ and $\alpha \in \mathbb{R}^+$, i.e. essentially integral. However, the latencies can be freely chosen from $\mathbb{R}^+$.

NOTE: This is ongoing work that has not been published. **Please DO NOT distribute this paper in public.**

This research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copy-right notation hereon.

II. Frame Graph

A. Graph Definition

Given a "step size", $\sigma \in \mathbb{R}^+$, we define the (infinite) weighted directed graph $H = (U, D)$ (with capacities $\gamma(d)$ on the edges $d \in D$) as follows:

Given an edge (v, w) define its "step length", $\bar{\lambda}(v, w)$, equal to $\lfloor \lambda(v, w)/\sigma \rfloor$.

The set of vertices, U, consists of the following elements:

1) For every $t \in \mathbb{N}$ and every $v \in V$ we have an element $v(t)$
2) For $t \in \mathbb{N}$, every $(v, w) \in E$, and every $i \leq \bar{\lambda}(v, w)$ we have an element $[v, w]_i(t)$.

The set of edges, D, is partitioned into frames $Z_1, Z_2, Z_3, ...$ where, for $t \in \mathbb{N}$, the frame Z_t is composed of the following: for every $(v, w) \in E$ we have the following directed edges, each of capacity $\beta(v, w)\sigma$ in Z_t:

1) An edge $(v(t), [v, w]_1(t-1))$.
2) For every $i < \bar{\lambda}(v, w)$ we have an edge $([v, w]_i(t), [v, w]_{i+1}(t-1))$.
3) An edge $([v, w]_{\bar{\lambda}(v,w)}(t), w(t-1))$.

Given an edge $e \in Z_s$ in a frame s, let $\uparrow_t(e)$ be the equivalent edge in Z_t (e.g. if $e = (v(s), [v, w]_1(s-1))$ then $\uparrow_t(e) := (v(t), [v, w]_1(t-1))$

We call H the "frame graph". We also define the "approximate graph" G' as equal to G but with, for every edge $(v, w) \in E$, the latency of (v, w) equal to $\sigma\bar{\lambda}(v, w)$. Note that G' becomes equal to G as $\sigma \to 0$ and hence we will now consider G' instead of G

B. Dynamic Flows

A "dynamic flow", F, on the frame graph H is a function $F : D \to \mathbb{R}^+$ such that $\sum_{u \in U:(u,v) \in D} F(u, v) = \sum_{w \in U:(v,w) \in D} F(v, w)$ for all $v \in U$ with $v \notin \bigcup_{t \in \mathbb{N}} \{\perp(t), \top(t)\}$.

So what is the intuition behind a dynamic flow F? In what follows, when we say "time frame t" (for $t \in \mathbb{N}$) we mean the time from $(t+1)\sigma$ seconds before the process (of routing bits) finishes to $t\sigma$ seconds before the process finishes. For $(v, w) \in E$ and $t \in \mathbb{N}$ the values of the dynamic flow represent the following:

1) $F(v(t), [v, w]_1(t-1))$ is the amount of data moving into the communication pipeline from v to w sometime during time frame t.
2) For every $i < \bar{\lambda}(v, w)$, $F([v, w]_i(t), [v, w]_{i+1}(t-1))$ is the amount of data that is $(i\sigma/\bar{\lambda}(v, w))^{\text{th}}$ down the

communication pipeline from v to w sometime during time frame t

3) $F([v,w]_{\bar{\lambda}(v,w)}(t), w(t-1))$ is the amount of data moving out of the communication pipeline from v to w sometime during time frame $t+1$

Given a dynamic flow F on the frame graph H and some $\tau \in \mathbb{N}$ we define the τ-value, F_τ, of the flow as $F_\tau := \sum_{t=1}^{\tau} \sum_{v \in U:(\perp(t),v)\in D} F(\perp t, v)$. Note that the τ-value is the total amount of bits sent from $\perp$ to $\top$ in the final $\tau\sigma$ seconds of the routing process using the dynamic flow F. So our goal is to find the minimum τ such that the maximum τ-value of any dynamic flow is Δ, this maximum dynamic flow then becoming the required routing.

Of course, given $\tau \in \mathbb{N}$ we can find a dynamic flow of maximum τ-value as follows: add vertices $\perp'$ and $\top'$ to U and for every $t \leq \tau$ add edges $(\perp', \perp(t))$ and $(\top(t), \top')$ to D. Then find a maximum flow from $\perp'$ to $\top'$ (which is the required dynamic flow). Interval bisection (using this process) on τ will then give us the minimum τ and its corresponding dynamic flow. However, for a large value of τ, finding the maximum flow from $\perp'$ to $\top'$ (as above) using standard techniques is very expensive (polynomial in τ). Hence, we must develop a faster algorithm to give the required routing.

C. Constructing a Dynamic Flow

We will use the notion of an augmenting path, a common concept in network flow algorithms. Given some weighted graph $A = (X, Y)$ with weight function $w : Y \to \mathbb{R}^+$ and vertices $a, b \in X$, a "path", P, in A from a to b is a sequence of vertices $S := \{a = x_1, x_2, x_3, ..., x_l = b\} \subseteq X$ along with a function $\psi_P : S \to \{-1, +1\}$ such that, for all $i < l$, if $\psi_P(x_i) = 1$ then $(x_i, x_{i+1}) \in Y$, else we have $(x_{i+1}, x_i) \in Y$. Given a function $f : Y \to \mathbb{R}^+$ with $f(e) \leq w(e)$ for all $e \in Y$, the f-capacity, $\phi_f(P)$, of such a path P is defined equal to:

$$\min\{w(x_i, x_{i+1}) - f(x_i, x_{i+1}) : \psi_P(x_i) = 1\} \quad (1)$$
$$\cup \{f(x_{i+1}, x_i) : \psi_P(x_i) = -1\} \quad (2)$$

Any path with non-zero f-capacity is called an f-augmenting path. Given such a path, P, we define the function $f \bigoplus P : Y \to \mathbb{R}^+$ by:

1) For all $i < l$ with $\psi_P(x_i) = 1$ we have $[f \bigoplus P](x_i, x_{i+1}) := f(x_i, x_{i+1}) + \phi_f(P)$
2) For all $i < l$ with $\psi_P(x_i) = -1$ we have $[f \bigoplus P](x_{i+1}, x_i) := f(x_{i+1}, x_i) - \phi_f(P)$
3) For all other edges $e \in Y$ we have $[f \bigoplus P](e) := f(e)$

Our algorithm (the correctness currently only being a conjecture) for constructing the required dynamic flow runs in rounds. For $i \in \mathbb{N}$, at the start of round i, we have a dynamic flow F_i, initialised such that $F_1(e) = 0$ for all $e \in D$. On round i we do the following:

1) Let s be equal to the minimum t such that that there exists an F_i-augmenting path from $\perp(t)$ to $\top(1)$. Let P be such a path. If there is no such F_i-augmenting path then quit the algorithm with F_i being the required dynamic flow.
2) Set $F' \leftarrow F_i \bigoplus P$
3) Write P as a sequence $\{x_1, x_2, x_3, ..., x_l\}$, of vertices in U
4) For all $i < l$:
   a) If $\psi_P(x_i) = 1$ set $e \leftarrow (x_i, x_{i+1})$. Else set $e \leftarrow (x_{i+1}, x_i)$
   b) Let r be such that $e \in Z_r$
   c) For all $t > r$ set $F'(\uparrow_t(e)) \leftarrow F'(e)$
5) Set $F_{i+1} \leftarrow F'$

III. Implicit Construction

Of course, the above explicit construction of the required dynamic flow is still computationally expensive (actually infinitely expensive). Hence, we now show how to do the above algorithm implicitly. Again, the correctness of this algorithm is based on conjectures. This algorithm uses only the graph G'.

The final scheduling (equivalent to a dynamic flow) will be represented as follows: Every edge $e \in E$ is assigned a sequence $\delta_e(1) < \delta_e(2) < ... < \delta_e(l) \in \mathbb{N}$, known as "change-points" and a sequence $\epsilon_e(1), \epsilon_e(2), ...\epsilon_e(l) \in \mathbb{R}^+$. Given these values the scheduling is defined as follows: for $t > \delta_e(l)$, $\epsilon_e(l)$ bits are passed down the communication pipeline e per second at timestep t (remember that this is $t\sigma$ seconds before the scheduling finishes). Given $i < l$ and $t \in \mathbb{N}$ with $\delta_e(i) \geq t > \delta_e(i-1)$, $\epsilon_e(i-1)$ bits are passed down the communication pipeline e per second at timestep t. The algorithm constructs these change-points with their respective values.

The algorithm is based on the following notion of "effective length": Given a path $P := \{v_1, v_2, ..., v_l\}$ in G', its "effective length" is defined equal to

$$\sum_{i<l:\psi_P(v_i)=1} \lambda(v_i, v_{i+1}) - \sum_{i<l:\psi_P(v_i)=-1} \lambda(v_{i+1}, v_i)$$

Again, the algorithm runs in rounds (corresponding exactly to the rounds of the previous, explicit, algorithm). The weights of the edges of graph G' (for the purpose of defining capacities of augmenting paths) are the bandwidths of the edges. On each round i we have a function $f_i : D \to \mathbb{R}^+$ (which is mathematically a flow on G') initialized as $f_i(e) \to 0$ for all $e \in E$. For each edge $e \in E$ we also store an integer $\mu(e)$ which is the number of change points found so far for edge e: initialised equal to 0. On round i the algorithm does the following:

1) Let P be an f_i-augmenting path in G' from $\perp$ to $\top$ of minimum effective length. If there is no such path then the algorithm terminates with the required scheduling.
2) Set $f_{i+1} \leftarrow f_i \bigoplus P$
3) Let $(\perp = v_1, v_2, ..., v_l = \top) := P$
4) For all $i < l$:
   a) If $\psi_P(v_i) = 1$ let $e \leftarrow (v_i, v_{i+1})$. Else let $e \leftarrow (v_{i+1}, v_i)$
   b) Let ζ be the effective length of the path $(v_i, v_{i+1}, v_{i+2}, ..., v_l = \top)$
   c) $\mu(e) \leftarrow \mu(e) + 1$
   d) $\delta_e(\mu(e)) \leftarrow \zeta$
   e) $\epsilon_e(\mu(e)) \leftarrow f_{i+1}(e)$

Note that the complexity of this algorithm doesn't depend on σ so we can limit $\sigma \to 0$ in order to solve the problem with the original graph G.