CLARINET: WAN-Aware Optimization for Analytics Queries

Raajay Viswanathan° *Ganesh Ananthanarayanan*† *Aditya Akella*°
°*University of Wisconsin-Madison* †*Microsoft*

Abstract

Recent work has made the case for geo-distributed analytics, where data collected and stored at multiple datacenters and edge sites world-wide is analyzed *in situ* to drive operational and management decisions. A key issue in such systems is ensuring low response times for analytics queries issued against geo-distributed data. A central determinant of response time is the query execution plan (QEP). Current query optimizers do not consider the network when deriving QEPs, which is a key drawback as the geo-distributed sites are connected via WAN links with heterogeneous and modest bandwidths, unlike intra-datacenter networks. We propose CLARINET, a novel WAN-aware query optimizer. Deriving a WAN-aware QEP requires working jointly with the execution layer of analytics frameworks that places tasks to sites and performs scheduling. We design efficient heuristic solutions in CLARINET to make such a joint decision on the QEP. Our experiments with a real prototype deployed across EC2 datacenters, and large-scale simulations using production workloads show that CLARINET improves query response times by $\geq$ 50% compared to state-of-the-art WAN-aware task placement and scheduling.

1 Introduction

Large organizations, such as Microsoft, Facebook, and Google each operate many 10s-100s of datacenters (DCs) and edge clusters worldwide [1, 5, 6, 13] where crucial services (e.g., chat/voice, social networking, and cloud-based storage) are hosted to provide low-latency access to (nearby) users. These sites routinely gather service data (e.g., end-user session logs) as well as server monitoring logs. Analyzing this *geo-distributed* data is important toward driving key operations and management tasks. Example analyses include querying server logs to maintain system health dashboards, querying session logs to aid server selection for video applications [15], and correlating network/server logs to detect attacks.

Recent work has shown that centrally aggregating and analyzing this data using frameworks such as Spark [48] can be slow, i.e., it cannot support the timeliness requirements of the applications above [24], and can cause wasteful use of the expensive wide-area network (WAN) bandwidth [35, 43, 36]. In contrast, executing the analytics queries *geo-distributedly* on the data stored *in-place* at the sites—an approach called geo-distributed analytics (GDA)—can result in faster query completions [35, 43].

GDA entails bringing *WAN-awareness* to data analytics frameworks. Prior work on GDA has shown how to make query execution (specifically, data and task placement) WAN-aware [43, 35, 36]. This paper makes a strong case for pushing WAN-awareness up the data analytics stack, into *query optimization*. While it can substantially lower GDA query completion times, it requires radical new approaches to query optimization, and rethinking the division of functionalities between query optimization and execution.

The query optimizer (QO) takes users' input query/script and determines an optimal *query execution plan* (QEP) from among many equivalent QEPs that differ in, e.g., their ordering of joins in the query. QOs in modern analytics frameworks [2, 7], largely use database technology developed over 30+ years of research. These QOs consider many factors (e.g., buffer cache and distribution of column values) but largely ignore the network because they were designed for a single-server setup. Some parallel databases considered the network, but they model the cost of *any* over-the-network access via a single parameter. This is less problematic within a DC where the network is high-bandwidth and homogeneous. Geo-distributed clusters, on the other hand, are connected by WAN links whose bandwidths are *heterogeneous* and *limited* (§2.1), varying by over 20×, because of differences in provisioning of WAN links as well as usage by different

(non-analytics) applications.

Given this heterogeneity, existing network-agnostic QOs can produce query plans that are far from optimal (§2.2). For example, QOs decide the ordering of multi-way joins purely based on the *size* of the intermediate outputs. However, this can lead to heavy data transfer over thin WAN links, thereby inflating completion times. Likewise, today's QOs optimize one query at a time; as such, when multiple queries are issued simultaneously, their individual QEPs can contend for the same WAN links. Thus, we need a new approach for WAN-aware multi-query optimization.

Arguably, because QOs are upper-most in analytics stacks, them being network-agnostic fundamentally limits the benefits from downstream advances in task placement/scheduling [21, 43, 35, 36]. However, as data analytics queries are DAGs of interconnected tasks, WAN-aware query planning itself has to be performed *in concert* with placement and scheduling of the queries' tasks and intermediate network transfers (§2.2), in contrast with most existing systems where these are conducted in isolation. This is because task placement impacts which WAN links are exercised by a given QEP, and scheduling impacts when they are exercised, both of which determine if the QEP is WAN-optimal. Unfortunately, formulating an optimal solution for such *multi-query network-aware joint query planning, placement, and scheduling* is computationally intractable.

We develop a novel heuristic for the above problem. First, we show how to compute the WAN-optimal QEP for a single query, which includes task placement and scheduling (§4). For tractability, our solution relies on reserving WAN links for scheduled (but yet to execute) tasks/transfers; however, we show that such link reservations lead to faster query completions in practice.

Given a batch of n queries, we order them based on their individually optimal QEPs' expected completion time; the QEP for the i^{th} query is chosen considering the WAN impact of the preceding $i-1$ queries. This mimics shortest-job first (SJF) order while allowing for cross-query optimization (§5.1). However, it results in bandwidth fragmentation (due to task dependencies), thereby hurting completion times. To overcome this, our final heuristic considers groups of $k \leq n$ queries from the above order and explores how to compact their schedules tightly in time, while obeying inter-task ordering (§5.2). The result is a cross-query schedule that veers from SJF but is closer to work-conserving, and offers low average completion times for GDA queries. We also extend the heuristic to accommodate fair treatment of queries, minimizing WAN bandwidth costs, and online query arrivals (§5.3).

We have built our solution into CLARINET, a WAN-aware QO for Hive [3]. Instead of introducing WAN-awareness inside existing QOs [2, 7], CLARINET is architecturally *outside* of them. We modify existing QOs to simply output all the functionally equivalent QEPs for a query, and CLARINET picks the best WAN-aware QEP per query, as well as task placement and scheduling which it provides as *hints* to the query execution layer. Our design allows any analytics system to take advantage of CLARINET with minimal changes.

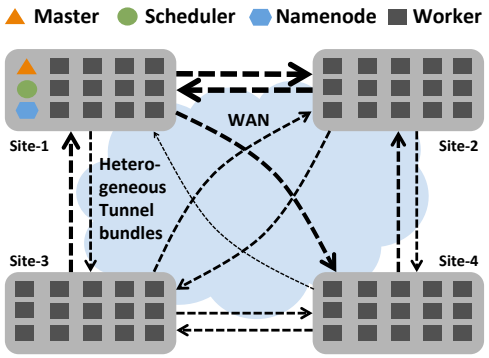


Figure 1: Architecture of GDA Systems

We deploy a CLARINET prototype across 10 regions on Amazon EC2, and evaluate it using realistic TPC-DS queries. We also conduct large scale trace-driven simulations using production workloads based on two large online service providers. Our evaluation shows that, compared to the baseline that uses network-agnostic QO and task placement, CLARINET can improve the average query performance by 60-80% percent in different settings. We find that CLARINET's joint query planning and task placement/scheduling doubles the benefits compared to state-of-the-art WAN-aware placement/scheduling.

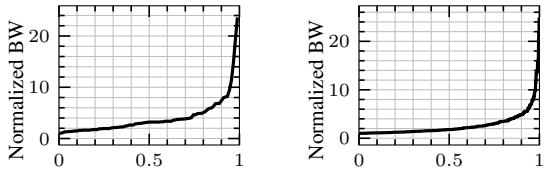
2 Background and Motivation

In this section, we first discuss the architectural details of GDA, focusing on WAN constraints. We then analyze how queries are handled in existing GDA systems.

2.1 Geo-Distributed Analytics

GDA Architecture: In GDA, there is a central *master* at one of the DCs/edge sites where queries—written, e.g., in SparkSQL [7], HiveQL [3], or Pig Latin [33]—are submitted. For every query, the QO at the master constructs an *optimized query execution plan* (QEP), essentially, a DAG of many interdependent *tasks*. A centralized scheduler places tasks in a QEP at nodes across different sites based on resource availability and schedules them based on task dependencies. [1]

[1] Typically, the task scheduler, the namenode of the distributed file system, and the master all run at the same site to reduce inter-process communication latencies between them. However, it is possible to distribute them across different processing sites.



(a) Amazon EC2 (b) MICROSOFT

Figure 2: Distribution of bandwidth between data processing sites for Amazon EC2 and a large OSP. The bandwidths reported are normalized with respect to the minimum observed. For Amazon EC2, the bandwidth between a pair of sites is obtained through active measurements using iperf. The minimum bandwidth obtained over 10-minute interval is taken as the guaranteed bandwidth. For MICROSOFT, we use the topology and traffic information from applications to compute the guaranteed bandwidth.

WAN Constraints: The sites are inter-connected by a WAN which we assume is optimized using MPLS-based [45] or software-defined traffic engineering [23, 20, 26]. In either case, end-to-end tunnels are established by the WAN manager for forwarding analytics traffic between all site-pairs. The WAN manager updates tunnel capacities in response to background traffic shifts. The running time of queries is typically lower than the interval between WAN configuration changes ($\sim$ 10 - 15 minutes [20, 23]); thus, we assume that the bandwidth between site-pairs remains constant for the duration of queries' execution. Thus, we can abstract the WAN as a *logical full mesh* with fixed bandwidth links (fig. 1).

However, available bandwidth between pairs of sites can differ significantly because of differences in physical topology and traffic matrix of non-analytics applications. Figure 2 highlights this variation between pairs of the 10 Amazon EC2 regions, and for the DCs operated by MICROSOFT. The ratio of the highest to lowest bandwidth is > 20. Also, the bandwidth is 1-2 orders of magnitude less than intra-DC bandwidth (e.g., the maximum inter-site bandwidth is 450Mbps between EC2 regions, where as intra-site bandwidth is 10Gbps). Thus, WAN bandwidth is highly constraining and a significant bottleneck for GDA, in contrast with intra-DC analytics.

2.2 Illustrative Examples for Drawbacks of Current GDA Query Processing

Given a query, its relational operators (e.g., SELECT, GROUPBY, JOIN, TABLESCAN, etc.) are transformed to individual "map" or "reduce" stages in the QEPs compiled by the QO. For example, SELECT and TABLESCAN are transformed to a map stage, whereas JOIN or GROUPBY are transformed to a reduce stage. If the input tables for these operators are partitioned and/or spread out across different sites, then the execution of downstream reduce stages (for JOIN or GROUPBY) will involve flow of data across the constrained WAN, limiting overall query performance.

Notation	Size
$\lvert\sigma_{\mathrm{A}}(*)\rvert, \lvert\sigma_{\mathrm{X}}(*)\rvert, \lvert\sigma_{Y\mid\mathrm{Z}}(*)\rvert$	200 GB, 160GB, 25 GB
$\lvert\sigma_{\mathrm{A\mid X}}(\mathrm{WS}) \bowtie \sigma_{\mathrm{A\mid X}}(\mathrm{SS})\rvert$	12 GB
$\lvert\sigma_{\mathrm{A\mid X}}(\mathrm{SS}) \bowtie \sigma_{\mathrm{A\mid X}}(\mathrm{CS})\rvert$	10 GB
$\lvert\sigma_{\mathrm{A\mid X}}(\mathrm{WS}) \bowtie \sigma_{\mathrm{A\mid X}}(\mathrm{CS})\rvert$	16 GB

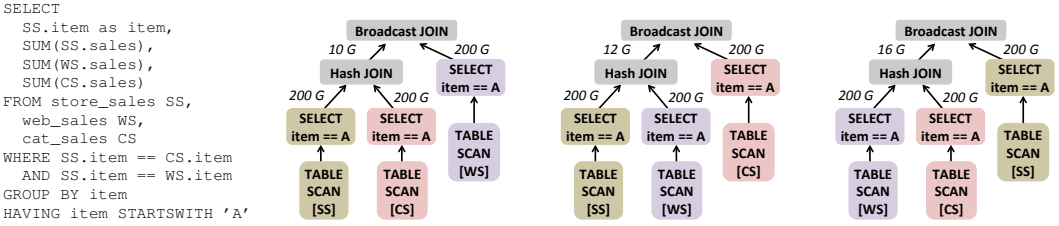
Table 1: Selectivity and join cardinality estimates

In what follows, we argue that because existing QOs do not account for such WAN constraints, their chosen QEP for a query may be sub-optimal. Because multiple queries can contend simultaneously for limited WAN bandwidth, QOs for GDA must account for two additional issues: (a) consider task placement and network transfer scheduling when determining a QEP's quality, and (b) plan for multiple queries at once.

Modern QOs employ a combination of heuristic techniques as well as cost based optimization (CBO) to determine the best execution plan for each query. Heuristic optimization leverages widely accepted techniques — e.g., predicate push down and partition pruning — for reducing query execution times.

CBO explores the space of all possible QEPs — e.g., those generated by considering alternate join ordering of tables — and chooses the one with the least cost. The cost of a QEP is based on a *cost model*, which captures the cost of accessing a single byte of data over a resource, and cardinality estimates of intermediate data based on individual and cross-table statistics, histograms of column values, etc. CBOs also account for a variety of factors, including availability of buffer cache and indexes. State-of-the-art technologies for accurate cardinality estimation and cost modeling have been developed over 30+ years of research. However, most have focused on single server systems which ignore the network as a factor in determining query performance [2, 10, 17]. Even parallel databases model the network as a "single pipe" which essentially assumes the entire network is *homogeneous* [34, 14, 25, 12, 32, 41, 38, 46, 49, 47, 28]. Such simple models are clearly insufficient to account for heterogeneous WAN bandwidths. Yet, this is clearly important in GDA (as shown in §2.1).

Importance of network-aware query optimization: Consider a three-way join, $\mathrm{Q_A}$: $\sigma_{\mathrm{A}}(\mathrm{WS}) \bowtie \sigma_{\mathrm{A}}(\mathrm{SS}) \bowtie \sigma_{\mathrm{A}}(\mathrm{CS})$ shown in fig. 3(a), that compares overall sales for a set of items (starting with 'A') across three different tables; the tables are spread across different sites inter-connected by a WAN (fig. 4). Q_{A} can be executed through three different QEPs (figs. 3(b)–3(d)); one each from three different join orders. Table 1 lists the sizes of different intermediate outputs.



(a) A sample SQL Query (b) QEP-1 (c) QEP-2 (d) QEP-3

Figure 3: An example SQL query and its different query execution plans. Each QEP corresponds to a different join order. Note how the selectivity predicate is pushed down to minimize the records processed during the joins.

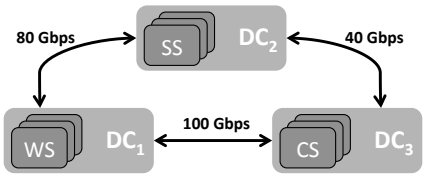


Figure 4: Three sites that are interconnected by bidirectional WAN links. Each site contains a unique table.

A network agnostic QO, or one that models the entire network by a single parameter, will pick QEP-1 since it has the least output cardinality after the first join. QEP-1 will take 20.5s: the first join, $(\sigma_{\text{A}}(\text{SS}) \bowtie \sigma_{\text{A}}(\text{CS}))$, will be implemented as a *hash join* since it involves large tables; by placing reducers uniformly across sites DC_2 & DC_3 the join involves 100GB of data transfer in either direction. Over a 40Gbps link, this transfer takes 20s. The second join will be implemented as a broadcast join since one of the tables is small. It involves transferring, 10GB of data spread across sites DC_2 & DC_3 to DC_1. The bottleneck is the transfer on the 80Gbps link which takes 0.5s.

Contrast this with QEP-3 that joins tables WS and CS first. Even though it has the highest cardinality for intermediate data, it might be advantageous because sites DC_1 & DC_3 have high bandwidth between them. By placing tasks uniformly between sites DC_1 & DC_3, the first join would take only 8s (100GB over 100Gbps link). The second join takes 1.6s (8GB over 40Gbps link). QEP-3 completes in 9.6s, or ≈53% faster.

Importance of considering placement in QEP selection: For each QEP, the exact pattern of traffic between the data processing sites is dependent not only on the nature of data flow between stages (e.g, scatter-gather, one-to-one) but also on the placement of tasks in each stage. Thus, placement must be taken into account in assessing QEP quality. Placement matters due to contention from currently running GDA queries.

Consider a scenario where an already running query is using the logical links, $\text{DC}_1 \rightarrow \text{DC}_3$ & $\text{DC}_2 \rightarrow \text{DC}_1$. Without control over task placement, a network-aware optimizer would choose QEP-1 for Q_A to avoid links already being utilized. Thus, its running time would be 20.5s. However, by choosing QEP-3 and placing all reduce tasks for the first join at DC_1, we can completely avoid $\text{DC}_1 \rightarrow \text{DC}_3$ & $\text{DC}_2 \rightarrow \text{DC}_1$ and finish in 17.6s.[2]

Importance of considering scheduling in QEP selection: Similar to placement, the impact of scheduling of network transfers should also be accounted for in assessing a QEP. Consider a query Q_X that is similar to Q_A in structure, but operates on a different slice of data, say, items starting with 'X'. Say Q_X arrives soon after two simple two-way joins, $\text{Q}_\text{Y} : \sigma_Z(WS) \bowtie \sigma_Z(CS)$ and $\text{Q}_\text{Z} : \sigma_Z(WS) \bowtie \sigma_Z(SS)$, start to execute. The selectivity information for input datasets of the queries is shown in Table 1. Being two-way joins, Q_Y and Q_Z have no choice of QEPs; they utilize the WAN bandwidth between DC_1 and DC_3 & DC_1 and DC_2, respectively. The joins take 5s each.

Without control over scheduling, QEP-1 is the best choice for executing Q_X (since it avoids the links used by Q_Y and Q_Z). Its completion time is 16.5s.[3] However, if we can control scheduling of queries, we can still choose QEP-3 for Q_X and delay its network transfers by 5s. The completion time is lowered to 13s.[4]

Multi-query optimization: QOs in modern stacks, e.g., Hive and SparkSQL, optimize each query individually. Resource contention among concurrent queries is potentially left to be resolved at the execution layer through scheduling and task placements. However, under scenarios where contention cannot be resolved, *jointly determining the QEP* for all queries provides better opportunities to avoid resource contention thereby resulting in lower query completion times. Classic multi-query database QOs [40] leverage efficient reuse of common sub-expressions across queries [39], shared scans [44] and sharing of buffer caches [19], but they do not model the network similar to single-query QOs.

[2] WS ⋈ CS takes 16s to transfer 200GB over 100Gbps, and the next join takes 1.6s to send 16GB over 80Gbps link.

[3] The first join of QEP-1 utilizes the bandwidth between DC_2 and DC_3 and takes 16s. The bottleneck for the second join–which starts after Q_Z completes—is the 80Gbps link and transferring 5Gb over it takes 0.5s.

[4] The first join of QEP-3 takes 6.4s (80GB between DC_1 and DC_3). The second join takes 1.6s to transfer 8GB from DC_3 to DC_2. With a delay of 5s (waiting for two-way join to finish) completion time is 13s.

Consider a case where Q_A and Q_X arrive concurrently. A network-aware query optimizer will choose the same QEP for both the queries resulting in contention for bandwidth on links between DC_1 & DC_3. The scheduler, to optimize for average completion time, will execute the shortest query first (Q_X in this case); the average running time will then be 12s. However, by choosing QEP-2 and QEP-3 for queries Q_X and Q_A respectively, we can completely avoid link contention and keep the average completion time at 9.4s (9.2 for Q_X and 9.6s for Q_A).

3 CLARINET's Design

Accomplishing multi-query network-aware plan selection and task placement/scheduling requires an analytics framework where a single entity is simultaneously responsible for both QEP selection and scheduling. Current big-data analytics stacks, however are highly modular with individual components being developed and operated independently. Realizing joint optimization in such a setting would require radical changes.

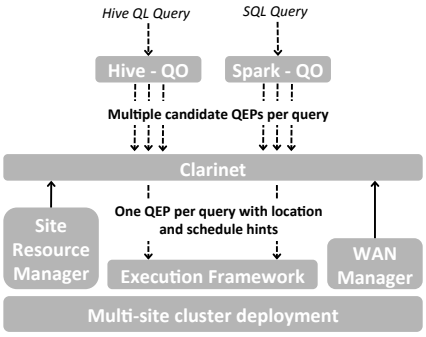


Figure 5: CLARINET's late-binding design

CLARINET's design (Figure 5) addresses this challenge via *late-binding*. In CLARINET, the QOs are modified to provide a set of *functionally equivalent* QEPs (QEP-Set)[5] to an intermediate *shim* layer. The shim layer (CLARINET) collects the QEP-Sets from multiple queries/QOs and computes a single, optimal QEP per query as well as location and scheduling *hints*, that it forwards to the execution framework.

Each node (operator) in a QEP, forwarded from QOs to CLARINET, is annotated with its output cardinality and parallelism as estimated by the QO. The cardinality represents the total amount of data transferred from the current operator to its successor operator. As this is data that will potentially be sent over the WAN, cardinality directly affects QEP selection. The operator parallelism decides the number of tasks to be spawned for each operator; the location and scheduling hints suggested by CLARINET correspond to location (at data center level) and start time for each task.

The late binding approach offers several **advantages**. First, given the complexity of QOs, modifying their internal cost model to account for the WAN is quite challenging. Also, QOs with widely different objectives (Calcite [2] vs Catalyst [10]) have to be modified individually. E.g., SparkSQL's QO [2] should factor availability of in-memory caches of RDDs [48] against WAN costs. By design, CLARINET requires no changes to a QO's internal cost model. Any QO that can provide multiple QEPs based on its current cost model can be made WAN-aware through CLARINET. Second, by introducing an intermediate layer, CLARINET alleviates (*i*) the analytics application (e.g., Hive) from making scheduling decisions and joint query optimization, (*ii*) the execution layer from making any network specific scheduling/placement decisions. This minimizes code changes to both the application and execution frameworks. Third, WAN awareness does not come at the cost of the existing query optimizations. An application can avoid the WAN from interfering with plan selection by exposing only its QO's chosen best QEP.

Problem statement, and assumptions: Given a set 'n' queries, CLARINET receives QEP-Sets, $QS_j, j \in [1, ..., n]$ from the QOs corresponding to each query. Among the exponentially many combinations, the objective is to select exactly one QEP from each QS_j along with task locations and schedules, such that the average query run time is minimized. Here, task locations determine the site at which tasks are executed, whereas the schedule determines the start times of each task.

For analytical tractability, we require that the tasks are scheduled such that network transfers on logical links between sites *do not temporally overlap* with one another. This allows us to accurately determine the duration of network transfers and reduce the QEP selection problem to a well studied job-shop scheduling problem. Such time multiplexing (or non-overlap) also has the advantage that resource sharing can be enforced through scheduling; (weighted) bandwidth sharing on the other hand requires additional per transfer rate-control on top of the rate control already enforced by the WAN manager. Crucially, the non-overlapped assumption does not affect the quality of the solution. This is because, as we prove below, any optimal schedule has an equivalent optimal non-overlapped schedule.

Theorem: A schedule, $\mathcal{S}$, of interdependent transfers over multiple network resources, where each transfer is allocated an arbitrary time-varying share of available

[5] A dynamic programming based QO will generate exponentially many query plans for each query. We limit the size of the QEP-Set by placing a bound (5 seconds) on time spent in exploring multiple query plans.

network bandwidth on a single resource, can be converted into an equivalent interruptible schedule, $\mathcal{N}$, such that no two transfers in $\mathcal{N}$ share a resource at any given point in time, and the completion time of a transfer in $\mathcal{N}$ is not greater than its completion time in $\mathcal{S}$.

Proof sketch: For a network transfer, f, on a resource, let $s(f)$ and $e(f)$ be the start and end times based on schedule, $\mathcal{S}$. For each resource, the start and end times of all its transfers can be viewed as its release time and deadline respectively. Converting $\mathcal{S}$ to $\mathcal{N}$, can be achieved by determining the *earliest deadline first* (EDF) schedule of flows for each resource independently. Given $s(f)$ and $e(f)$, an EDF schedule is feasible since S is a complete schedule. For a detailed proof, refer [42].

We simplify further and focus on obtaining *non-interruptible* transfer schedules, because, implementing interruptible transfers requires significant changes to query execution. However, such schedules do not permit perfect "packing" of transfers across links. The resulting fragmentation of link capacity delays scheduling of network transfers, and inflates completion times. Essentially, CLARINET incorporates a clever approach—which we develop gradually in the next two sections—that systematically combats such resource fragmentation and optimizes average query completion times.

In sum, our simplifying assumptions do not impact CLARINET's effectiveness. However, even with these assumptions, computing the best cross-query QEPs along with task placements and schedules is NP-hard. In fact, the problem is hard even for a single query [30, 31].

Our approach is as follows: we start with an effective heuristic for the best single-query QEP, that decouples placement and scheduling (§4). We then use this to gradually develop our multi-query heuristic (§5).

4 Single Query WAN-Awareness

At a high level, WAN-aware QEP selection for a single query proceeds as follows: for every QEP in the query's QEP-Set, we determine the placement and schedule of tasks such that its running time is minimized. The QEP with the shortest running time is then selected. Because of inherent hardness of joint placement and scheduling of DAGs [30, 31], CLARINET's approach is to decouple them, as described next.

4.1 Assigning Locations to Tasks in a QEP

Given a QEP, for tasks with no dependencies (e.g., map tasks) we use the common approach of "site-locality", i.e., their locations are the same as the location of their input data. The placement of intermediate (reduce) tasks is decided based on the amount and location of intermediate data generated by their parents, along with the bandwidths at the sites.

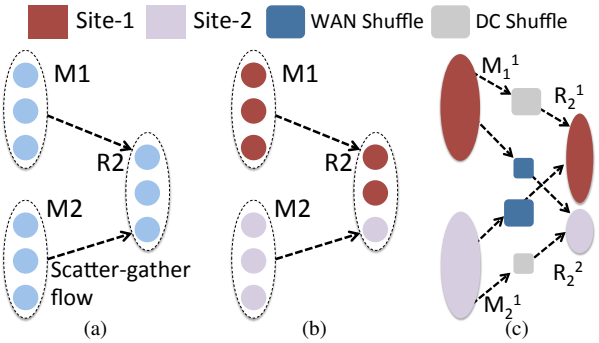


Figure 6: (a) shows a simple MR job with 3 tasks in each stage. (b) shows the same job with placement information (color-coded) for all the tasks. (c) shows the corresponding augmented DAG with tasks in the same stage and location coalesced to one. M_*^* and R_*^* are sub-stages in the augmented DAG. Shuffle tasks representing network transfer between tasks at different locations are shown explicitly.

We decide the task placements for a QEP iteratively for each of its stages in topological order. Since the query plan specifies a partial ordering between dependent stages, stages with no order among them can be simultaneously scheduled. To ensure that the placement decision for these stages takes into account other stages' decisions, we *reserve* a block of time on logical links for network transfers (consistent with our non-overlapped assumption).

Formulation: The optimal task placement for a stage is obtained by solving a linear program which takes as input the following: *(i)* the distribution of output data (D_ℓ) from the predecessor stages across sites (ℓ), *(ii)* the inter-site WAN bandwidths ($B_{\ell_1}^{\ell_2}$), and *(iii)* the length of time ($\tau_{\ell_1}^{\ell_2}$) for which stages (which do not have ordering with respect to current stage) have reserved inter-site links.[6] The best distribution of tasks (r_ℓ) across sites is obtained by solving the following problem:

$$\min_{r} \quad \sum_{\ell_1,\ell_2} \frac{D_{\ell_1} r_{\ell_2}}{B_{\ell_1}^{\ell_2}} + \tau_{\ell_1}^{\ell_2} \tag{1a}$$

$$\text{such that} \quad r_{\ell_2} \geq 0 \tag{1b}$$

$$\sum_{\ell_2} r_{\ell_2} = 1 \tag{1c}$$

Once the locations of the reducers are fixed, we use the resulting traffic pattern to update the durations for which resources are blocked for later stages. E.g., between ℓ_1 and ℓ_2 we increment the duration, $\tau_{\ell_1}^{\ell_2}$, by $\frac{D_{\ell_1} r_{\ell_2}}{b_{\ell_1}^{\ell_2}}$.

4.2 Scheduling tasks in a QEP

In contrast to scheduling within a DC, scheduling a QEP in a geo-distributed setting involves scheduling both the compute phase of a task and the transfer

[6] ℓ, ℓ_1, ℓ_2 are indices over the set of data processing sites

of input data from remote sites. To explicitly model these network transfers, we augment the DAG of tasks representing the QEP to include vertices (called *shuffle tasks*) corresponding to network transfers; fig. 6 shows an example. We assume that the compute phase of a task can only start after all its inputs are available at the site. Further, since tasks of a stage that are executed at the same site exercise the same network resource, we coalesce them into a *sub-stage*. This reduces the number of entities that need to be scheduled and the overall scheduling complexity.

We formulate the scheduling as a binary integer linear program that takes as input the following: *(i)* the coalesced DAG augmented with shuffle tasks, henceforth called *augmented-DAG*, which captures dependencies among tasks, and *(ii)* the duration of compute and shuffle tasks. The duration of a compute task is same as the expected running time of the task in a intra-DC setting. The duration of shuffle between sites is estimated as the ratio of data transferred to the WAN bandwidth between the sites. The objective is to determine the optimal start times for all the tasks in the augmented-DAG such that the overall execution time of the QEP is minimized.

Formulation: Let c^i be the augmented-DAG of the i-th QEP in the QEP-Set for the query. Let V^i represent the set of vertices in c^i and $\leq^i$ represent the partial order between them. The start times of the vertices ($s(.)$) should obey the partial order $\leq^i$. Thus, for each pair of ordered vertices, $(u, v) \in \leq^i$, belonging to c^i,

$$s(v) \geq s(u) + d(u) \tag{2}$$

where, $d(.)$ represents the duration of vertices.

We incorporate non-overlapping of flows on network links in our scheduling problem by imposing:

$$s(v) \geq s(u) + d(u) - N(1 - z_{uv}) \tag{3a}$$
$$s(u) \geq s(v) + d(v) - N(z_{uv}) \tag{3b}$$

where u and v are shuffle tasks that contend for the same network link, and z_{uv} indicates v is executed after u; N is large constant. When $z_{uv} = 1$, then Equation 3a ensures that the start time of vertex v is greater than the completion time of vertex u; Equation 3b remains void, since it is satisfied trivially. When $z_{uv} = 0$, the conditions invert. Equations (3a) and (3b) are enforced for all links.

The completion time (Φ^i) of the i-th QEP, is given by:

$$\Phi^i := \max_{u \in V^i} \quad s(u) + d(u) \tag{4}$$

where, u is any vertex in c^i. We solve the program for all the QEPs in the QEP-Set. The one with the smallest duration is chosen to be executed.

Handling currently running queries: We "reserve" network links for tasks already placed and scheduled. Therefore, while computing the best schedule for a QEP, we have to factor in currently running queries that block resources. We add constraints to the above formulation in order to accommodate these currently running queries.

Let $B(r)$ be a set of time intervals for which existing queries block resource r. Let $low(b)$ and $high(b)$ represent the lower and upper bound of an interval $b \in B(r)$. For every vertex u using a network link, we include these two constraints:

$$s(u) \geq high(b) - N(1 - x_{ub}) \tag{5a}$$
$$low(b) \geq s(u) + d(u) - N(x_{ub}) \tag{5b}$$

where x_{ub} is a binary indicator denoting that u is scheduled after the interval b. Like eqs. (3a) and (3b), these constraints kick in alternatively ensuring the transfers do not overlap with intervals that are reserved.

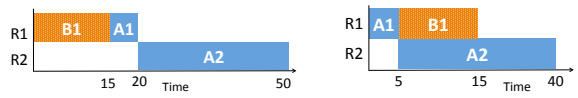
5 Multiple Contending Queries

In this section, we build upon the solution in §4 to solve the problem statement outlined in §3 for a workload of 'n' (>1) queries that arrive simultaneously and compete with each other for the inter-site WAN links.

In §5.1, we first provide a strawman algorithm that emulates shortest-job first (SJF), and iteratively determines the QEP, placement, and schedule for each of the 'n' queries. Unfortunately, the strawman algorithm results in a schedule with link resources being fallow close to 22% of the time (ref. Figure 12(a)) due to resource fragmentation. In §5.2, we present a novel heuristic that builds on the strawman and minimizes resource fragmentation; it combats fragmentation by carefully packing flows from k ($\leq n$) queries at a time from the schedule determined by the strawman. We discuss several enhancements in §5.3.

5.1 Strawman Iterative QEP Selection

Our strawman heuristic is based on shortest-job first (SJF) scheduling. We pick this because it is a well understood scheduling discipline that is typically used to minimize average completion times. Our strawman functions iteratively. In every iteration, we pick the QEP and determine the schedule for exactly one query as follows. For each QEP belonging to the QEP-Set of unscheduled queries, we calculate its duration, placement, and schedule of tasks (using techniques in §4). We then pick the QEP (and thus the query) with *shortest duration* among all the QEPs considered; we do not consider this query for future iterations. At the end of each iteration, we reserve resources required by the QEP chosen. By doing so, we ensure that the running time for the query is not affected by queries considered in future iterations. Further, it ensures that choice of QEPs in future iterations account for the current query's WAN impact, thereby enabling cross-query planning.



(a) SJF schedule (b) Better Schedule

Figure 7: Example highlighting fragmentation of resources with SJF heuristic. Tasks A1 and A2 belong to Job A; A2 is dependent on A1. Job B has only one task, B1. A1 and B1 use resource R1, A2 uses resource R2. In the SJF schedule, resource R2 remains idle until $t = 20$s.

5.2 Final Heuristic to Combat Resource Fragmentation

The above iterative heuristic is not ideal because it can cause links to remain fallow sometimes, *even if* there are other flows which could use those links. See Figure 7; as shown, fragmentation arises because jobs need multiple resources (multiple links in this case) and because of dependencies across tasks. What this shows is that vanilla SJF is not ideal for minimizing average completion times in our setting. If not controlled, underutilization of link resource can delay query completions arbitrarily.

We address this by modifying the above SJF strawman to use a knob (k) that reduces resource fragmentation. The knob allows us to deviate away from the iteratively computed schedule towards a schedule with low fragmentation of resources in a controlled manner. We start with the solution obtained from the iterative algorithm described in §5.1. The solution determines the following: *(i)* $\mathcal{O}$, a total ordering over the queries, based on their time of completion, and *(ii)* the mapping of inter-site flows to network resources (obtained from the choice of QEP and task placement). With this information, our final heuristic creates a constrained schedule as follows:

We maintain a dynamic set, $\mathcal{D}$, consisting of k-shortest queries (based on ordering $\mathcal{O}$), for which at least one task is not yet scheduled. Whenever a flow belonging to a query in $\mathcal{D}$ is available (i.e., at the time when all its predecessors have completed), and the resource it needs is free, we immediately schedule the flow on the resource rather than wait for its start time based on the iterative schedule. If multiple flows meet the criteria, we break ties in favor of short duration flows. When all tasks for a query are scheduled, it is dropped from the dynamic set and a new query is added.

When $k = 1$, only flows belonging to the shortest query can be moved ahead; thus the resulting schedule will be close to the strawman's. When k equals the total number of concurrent queries N, the resulting schedule will have no fallow links (note that the query completion times and the ordering of flows on a resource will be different from that computed using our iterative SJF algorithm). But, it may not offer good performance. This happens because, at high values of k, the initial stages (mappers) of the k QEPs are scheduled first, as they are available immediately. Thus, resources are indiscriminately blocked for later stages for *all* k QEPs, resulting in an increase in average completion times. We evaluate this effect in §7, and show the optimal average completion time benefits of an ideal "sweet-spot value" of k.

Note that in this heuristic, only the schedule is altered; task the placement and the QEP remains the same.

5.3 Enhancements

Fairness: Our heuristic can lead to long queries' start times being pushed significantly to favor shorter running queries. This is not acceptable if the long queries are initiated by different applications that require performance guarantees. To mitigate this bias, we adopt an approach similar to [22]. Essentially we want to ensure that the running time of a query, Q_j, is bounded by $d_j = n \times dur_j$, where n is the number of simultaneously running queries, dur_j is the standalone run time of the query without contention, and d_j denotes the calculated deadline for each query.

Then, we adapt the heuristic in §5.2 as follows. We sort queries in descending order based on a "proximity score"; this score determines how close a query is to its deadline and is obtained as:

$$Proximity_j(t) = 1 - \frac{d_j - t}{d_j} \tag{6}$$

where t is the time at which the dynamic set (§5.2) is updated (upon completion of a query). We pick the top ϵM queries in this sorted order and call them $\mathcal{H}$. Here, ϵ ($0 < \epsilon \leq 1$) is a fairness control knob and M is the number of queries with at least one task not yet scheduled. The dynamic set $\mathcal{D}$ (from §5.2) is then obtained by picking the shortest-k queries from $\mathcal{H}$. If $k > |\mathcal{H}|$, then $\mathcal{D} = \mathcal{H}$. By doing so, we block the tasks of queries that are far from their deadline from being scheduled and prefer those closer to their deadline.

When $\epsilon = 1$, $\mathcal{H}$ contains all the remaining queries and the heuristic is identical to the one in §5.2. When $\epsilon \to 0$, $\mathcal{D}$ contains only queries with highest proximity to fair-share deadlines; thus, offering maximum fairness.

WAN utilization: By favoring QEPs and task placement that result in smaller completion times, CLARINET implicitly reduces the WAN usage. However, unlike recent work [43], CLARINET cannot provide explicit guarantees on WAN usage. To explicitly control WAN usage, we filter from the QEP-Set of all queries those QEPs whose best (in terms of WAN use) task placement results in inter-site WAN usage exceeding a threshold, β. With a limited set of QEPs per query, we

then apply techniques in §5.1 and §5.2 for scheduling the transfers.

Online arrivals: We assumed so far that the set of n queries arrive simultaneously. We now extend the heuristic in §5.2 to support online query arrivals. Upon arrival of a new query, we recompute the QEP choice, task placement, and schedule for the current query together with all previous queries for which none of the tasks have started executing. Doing so might alter the QEP and schedule for prior, as yet unexecuted queries based on new information. Changing the QEP for already executing queries would incur wastage of resources; CLARINET does not alter the QEP for those queries.

6 Implementation

We build CLARINET as a stand-alone module that can interface with Hive [3] at the application level and Tez [4] at the execution framework level. We modified Hive and Tez to interface with CLARINET as follows:

Modifications to Hive/Calcite: Hive internally uses the Apache Calcite [2] library as a CBO. Calcite offers two types of QOs: *(i) HepPlanner*, which is a greedy CBO, and *(ii) VolcanoPlanner*, a dynamic programming-based CBO [16], which enumerates all possible QEPs for a query. By default, Hive uses the HepPlanner, but since it does not explore all possible QEPs, we modify Hive to interface with VolcanoPlanner. We further modify VolcanoPlanner to return the operator trees (OPT) representing multiple join orders along with the estimated cardinality (in bytes) for each operator, for each input query. All the OPTs are then compiled to corresponding QEPs by applying heuristic physical layer optimizations like partition pruning, field trimming, etc. The QEPs together constitute the QEP-Set for the query. We find that a typical TPC-DS [8] query has tens of QEPs in its QEP-Set. Each QEP is also annotated with the estimate of intermediate data for each stage; this is used by CLARINET to estimate network transfer times.

Modifications to Tez: CLARINET interfaces with Tez by providing *hints* regarding placement locations and start times for individual tasks. We modify Tez's DAG scheduler to schedule tasks based on these inputs. If a task becomes available before its scheduled start time, we hold it back and schedule it for execution later; a task is never held back beyond its scheduled start time.

Scheduling non-overlapped transfers: CLARINET employs a schedule that requires non-overlap of flows between two sites. Consider the simple MapReduce job similar to one in fig. 6(a). If tasks from two map stages (say, M_1^1 and M_2^1) are executed at the same location, then the transfer of their intermediate data to any downstream task (say, R_1^1) happens in an overlapped fashion; i.e., when R_1^1 starts executing, it reads data written by both M_1^1 and M_2^1 simultaneously. To enforce non-overlapped transfers by controlling task schedule, we introduce *relay* stages in the QEP (stages F and G in fig. 8(b)). The task in a relay stage does not process data; it reads remote data and writes it locally. Its parallelism and locations are identical to the corresponding reducer stage. By specifying start times of tasks (F_1^1 and G_1^1 in fig. 8(b)) in the relay stage, CLARINET explicitly determines start times of inter-stage shuffles and can ensure they happen in a non-overlapped fashion.

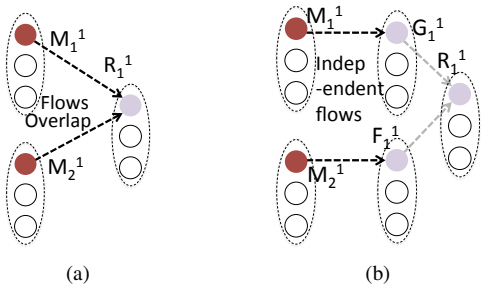


Figure 8: Modification of QEPs forwarded to execution framework by adding relay stages (F and G). Relay stages ensure network transfers fully utilize bandwidth and can be scheduled in a non-overlapped fashion. Here, map tasks M_1^1 and M_2^1 are executed in the same site, whereas reducer task, R_1^1 is executed in a different site. Relay tasks, F_1^1 and G_1^1 are co-located with R_1^1.

7 Evaluation

We experimentally evaluate CLARINET in realistic settings and against state-of-the-art GDA techniques. We evaluate CLARINET first in a real GDA deployment over 10 Amazon EC2 DCs. We use the standard TPC-DS [8] workload for benchmarking. For evaluating CLARINET at a large scale, we also use traces from analytics queries executed on two OSPs' production clusters. We simulate a GDA setup spread across tens of DCs and executing 1000's of queries. By default, we run CLARINET without the fairness enhancement.

The *de-facto* way in which queries are executed in a Hive-atop-Tez deployment is used as the baseline for comparison. Specifically, query selection and task placement are both network agnostic; here the QEP is selected by Hive's default QO and the reducers are placed uniformly across sites where input data is present. Since we are interested in reducing average completion time, we use our shortest query first heuristic (SJF; §5.1) to schedule the tasks belonging to multiple queries. We call the baseline HIVE+.[7]

Prior work [35] has shown that centrally aggregating raw data to one DC is wasteful. However, they only

[7] The '+' in HIVE+ indicates that the SJF heuristic is used for multiple queries. In a normal deployment,concurrent queries will arbitrarily share WAN bandwidth thereby delaying completion time for all.

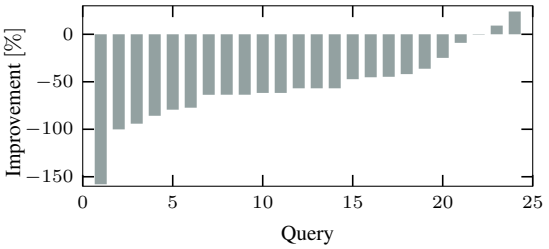


Figure 9: Percentage reduction in running time of HIVESINGLEDC w.r.t. HIVE+ for TPC-DS queries (sorted by increasing gains). The negative gains indicate running times of queries with HIVESINGLEDC is greater than HIVE+.

consider the case where raw input data is centrally aggregated; it is possible to reduce the amount of data sent over the WAN by suitably processing/filtering raw input data. For completeness, we evaluate our baseline (HIVE+) against an alternative that centrally aggregates data after pre-processing; we call this alternative, HIVESINGLEDC. In our implementation, HIVESINGLEDC uses the default QEP chosen by Hive's QO; the map tasks which process (filter) the raw input data are co-located with the data and all reduce tasks are placed in the DC with maximum intermediate data after map stages.

We also study HIVE-IR+ in simulation. HIVE-IR+ uses the QEP chosen by Hive but decisions on placement and scheduling are made using algorithms described in §4 and §5. The IR in HIVE-IR+ stands for Iridium [35], a state-of-the-art scheme for WAN-aware data/task placement. CLARINET's task placement is similar to Iridium's [35]. However, we use the "+" suffix since Iridium only does network-aware data/task placement, whereas HIVE-IR+ also does network-aware transfer scheduling. Comparing CLARINET and HIVE-IR+ highlights the importance of doing QEP selection along with WAN-aware task placement and transfer scheduling. We measure the improvements of CLARINET and HIVE-IR+ in terms of percentage reduction in average query run time compared to HIVE+.

7.1 Testbed Deployment Results

Deployment Setup and Workload: We spin up 5 server instances each with 40 vCPUs (2.4 GHz Intel Xeon Processors) and 160GB RAM in all 10 EC2 regions. We deploy HDFS+YARN across all the instances; a single server in one of the regions functions as the HDFS namenode and the YARN resource manager. The connectivity between different sites is through the public Internet; naturally available bandwidth (see fig. 2) acts as the constrained resource. To avoid disk read/write bottlenecks, we store all the intermediate data in memory; this also aligns with recent trends toward in-memory analytics [48, 27]. We use TPC-DS queries on datasets at different scales (10, 50, 100, 500) for our evaluation. Our workload is generated by randomly choosing the queries and the scale of data. The input tables are randomly spread across the different geographical regions, similar to prior studies [35, 43].

Comparison with single DC execution model: Figure 9 compares running times of individual TPC-DS queries using HIVESINGLEDC and HIVE+. For only 2 of the 24 different queries that we evaluated, HIVESINGLEDC has a smaller running time than HIVE+; further, HIVESINGLEDC can be up to four times slower ($0.25\times$) than HIVE+.

Upon closer investigation, we find that for the queries where HIVESINGLEDC is faster, the distribution of the largest input table was skewed; 70% of the input data was in one DC. Thus, for such cases, HIVESINGLEDC requires only 30% of the mapper outputs and none of the reducer outputs to be transferred across the WAN.

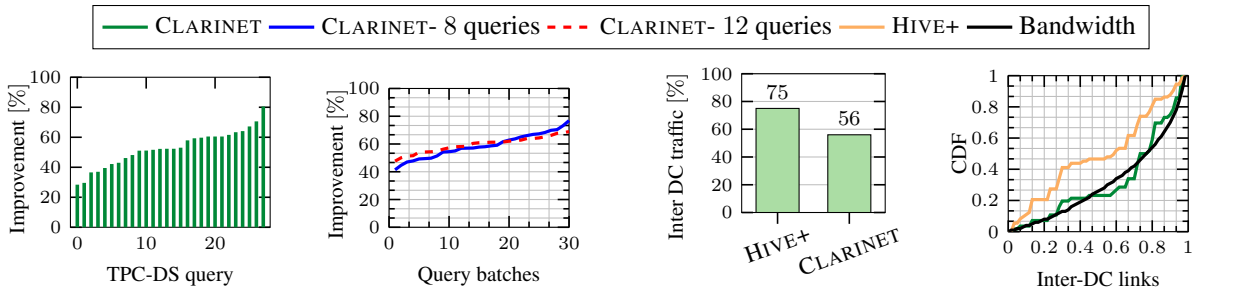
Overall, the distributed execution model effectively utilizes the total WAN bandwidth when the input data is spread across multiple DCs. However, when the input data is skewed, placing all the reducers in one DC is advantageous. Thus, in all further experiments, we also consider a task placement strategy where all the tasks are placed in the DC with the largest input data in addition to the placement approaches discussed in §4.1. CLARINET's design and the iterative heuristic described in §5.1 easily accommodate multiple task placement strategies for each QEP.

Clarinet performance: Figure 10(a), shows the run time reduction of CLARINET compared to HIVE+ for TPC-DS queries when run individually. We can see that network-aware QEP selection, task placement, and scheduling results in at least a 20% reduction (or 1.25x speedup) in query run time; the gains can be as high as 80% (5x) for some of the queries. For 75% of the queries, CLARINET chooses an alternate QEP than the one chosen by default in Hive (not shown). This highlights the importance of network-aware QEP selection even for single queries.

Figure 10(b) shows the gains when multiple TPC-DS queries of different scales are run simultaneously; we report results over 30 randomly chosen batches of TPC-DS queries with 8 and 12 queries in a batch. More than 40% (1.66x) gains are observed in all the batches. On average, we see a $\approx 60\%$ reduction or 2.5x speedup, higher than the single query case (45% on average).

By placing the reducers randomly across different geographical regions, HIVE+ transfers 75% (Figure 10(c)) of the total intermediate data between EC2 DCs. Since, the inter-region bandwidth is limited, this leads to longer running times. CLARINET on the other hand, transfers only half of the intermediate data between DCs.

Figure 10(d) shows the distribution of bandwidth and intermediate data across the inter-site links for a



(a) Percentage reduction in running times of CLARINET w.r.t. HIVE+ for 29 individual TPC-DS queries. The queries are sorted based on the observed gains.

(b) Percentage reduction in average completion times of CLARINET w.r.t. HIVE+ when batches of 8 / 12 randomly chosen TPC-DS queries of different scales are executed simultaneously. The batches are sorted based on the observed gains.

(c) Comparison of HIVE+ and CLARINET w.r.t. intermediate data sent over the WAN as a percentage of the total intermediate data. The values are measured over a single run with 12 simultaneous queries.

(d) Comparison of bandwidth and intermediate data distribution across a subset of pairwise logical links between the DCs for a single batch of 12 queries. We ignore all links that are unused by both CLARINET and HIVE+.

Figure 10: Results from a real CLARINET deployment across Amazon EC2 datacenters.

single run with 12 simultaneously running queries. The difference between intermediate data and bandwidth distribution is greater for HIVE+ when compared to CLARINET. For example, HIVE+ transfers 45% of its intermediate data over logical links that account for only 20% of the bandwidth. In comparison, CLARINET transfers only 20% of its load on 20% of the bandwidth. By considering multiple candidate QEPs for each query and by controlling task placement and scheduling, CLARINET is able to match intermediate data to available bandwidth across different links. Since HIVE+ does not have alternate choices of QEP and task placement/schedule, it tends to put more load on some links and no load on others.

Multi-query optimization: To quantify the need for multi-query optimization in the geo-distributed setting, we measure how the QEPs chosen for queries when run in a joint manner differ from the QEPs chosen when run individually. For 60% of the queries when run with 8 or 12 queries concurrently, the QEP of choice in CLARINET differs from the one chosen when the queries are run individually. As an illustrative example of CLARINET's cross-query behavior, consider TPC-DS query 7; it involves a five-way join of a fact table with 4 other dimension tables one of which is fairly large. Thus, when run by itself, CLARINET never joins the fact table with a large dimension table (even though they are located in DCs within a continent) to avoid costly WAN transfer. However, in 5 out of 6 batches when Query 7 runs simultaneously with other queries that load links behind the preferred dimension table, CLARINET forces Query 7 to join large tables upfront.

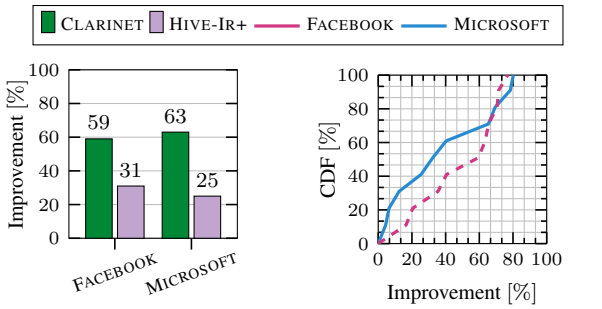
Resource Fragmentation: For a single run with 12 simultaneously running queries, we compute the duration for which inter-DC links remain idle. A resource is *idle* if a task is available to run, but is not scheduled for execution. For CLARINET, *the links are fallow only for 3% of the time*, which is minimal. Our larger scale simulation results confirm reduction in resource fragmentation imposed by our approach.

Optimization overhead: We also measured the time CLARINET spends in optimizing the query plan. After parallelizing the evaluation of each candidate query for every iteration, we see that CLARINET spends less than 1s (on an average) per iteration. For optimizing 12 queries in 30 different batches, CLARINET takes a maximum of 15s; the median optimization time is 8s for a batch of 12 queries. Relative to query execution times (tens of minutes), the optimization overhead of CLARINET is acceptable in practice.

7.2 Simulation Results

Trace driven Simulator: For large-scale experiments, with 50 sites and thousands of queries, we evaluate CLARINET through a trace-driven simulation *based on* production traces obtained from analytics clusters of two large OSPs, FACEBOOK and MICROSOFT. These traces contain information on query arrival times, input data/intermediate data size for each query, data locations, QEP structure etc., for 350K and 600K jobs respectively. Please refer to [37, 18] for more details about the workloads.

Unfortunately, we do not have logs from the query optimizer that generated the QEP, and hence do not have information regarding alternate QEPs. To overcome this, we use QEPs generated from TPC-DS queries superimposed with information on input table size and intermediate data size from the traces. Thus, every job in the trace is replaced by a randomly chosen TPC-DS query. The TPC-DS input tables acquire the distribution and location characteristics of input data for the corresponding job in the trace. Thus, our workload



(a) Percentage reduction in average running times of CLARINET and HIVE-IR+ w.r.t HIVE+.

(b) CDF of the per-query gains of CLARINET using FACEBOOK and MICROSOFT traces.

Figure 11: Overall gains of CLARINET and HIVE-IR+ w.r.t HIVE+ as measured in the simulator using FACEBOOK and MICROSOFT production traces

has similar load and data distributions as the production traces but using query plan options from the TPC-DS benchmark.

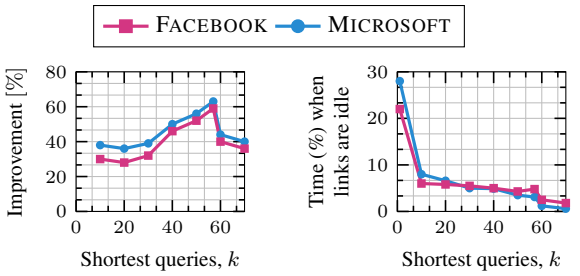
Queries arrive in batches of a few hundred; results for other batch sizes are similar. We impose a logical full mesh topology with the bandwidth between each pair of sites chosen randomly from $[100\text{Mb/s} - 5\text{Gbps}]$.

Figure 11(a) shows the reduction in average running time for CLARINET and HIVE-IR+ when compared to HIVE+ for both the production traces. Compared to the HIVE+ base line, CLARINET improves the average query completion time by 60%, or a 2.5× speedup.

CLARINET offers 28 and 38 percentage points improvement over HIVE-IR+ for FACEBOOK and MICROSOFT traces, respectively. This translates, respectively, to 1.75× and 2× speedup relative to HIVE-IR+. These additional gains come from choosing better QEPs.

For a network topology with higher bandwidths and low variation (drawn from $[10\text{Gbps} - 50\text{Gbps}]$), CLARINET has 47% and 52% reduction in run time for FACEBOOK and MICROSOFT traces, respectively, relative to HIVE+. Higher bandwidth implies overall smaller running times even for a WAN-agnostic system like HIVE+. Even under such a scenario CLARINET offers a 2× improvement.

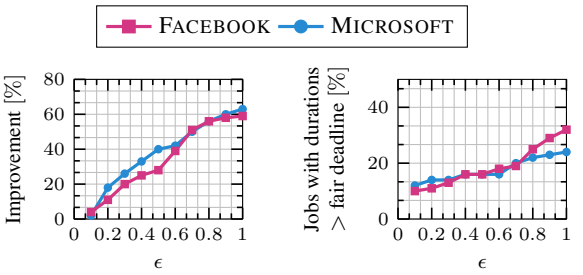
Figure 11(b) plots the distribution of CLARINET's gains w.r.t HIVE+. Note that CLARINET does not increase the running time for any query. However, the distribution has a heavy tail; some queries have moderate improvement but others have substantial improvement. The variation is especially prominent in MICROSOFT traces, where approximately 38% of the queries have less than 20% (1.25x) improvement and 20% of the queries have greater than 70% improvement (or 3x speedup). In §7.4, we present an in-depth analysis of performance improvement for different classes of queries.



(a) Variation of performance with k, for shortest-k heuristic

(b) Reduction in resource fragmentation with increasing k

Figure 12: Performance of our overall heuristic as a function of k.



(a) Percentage reduction in average query run times relative to HIVE+

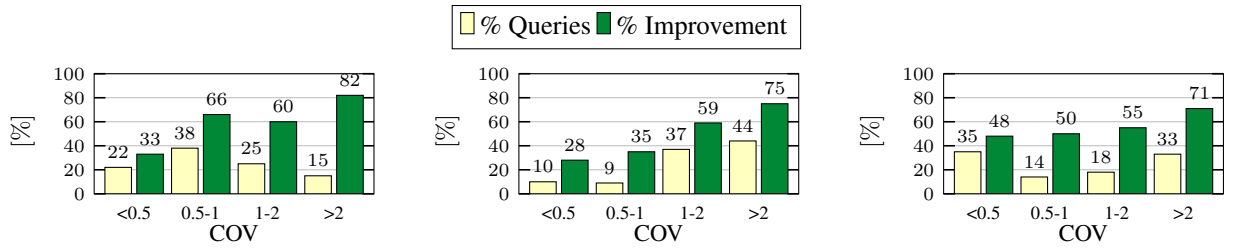
(b) Percentage of jobs that do not meet their fair deadline

Figure 13: Variation of performance and fairness metric with respect to ϵ

7.3 CLARINET's heuristics and design decisions

Next, we explore the effectiveness of key CLARINET design decisions in simulation.

Effectiveness in Combating Resource Fragmentation: Recall from §5.2 that our approach to combat resource fragmentation is to allow network transfers from top-k shortest queries to be scheduled if resources are fallow. Figure 12(a) plots the variation of overall runtime reduction for different values of k. For $k = 1$, CLARINET does vanilla SJF scheduling. As we increase k, the gain increases, peaks at $k = 57$ for both the FACEBOOK and MICROSOFT traces, and then decreases. The vanilla shortest job is considerably worse than choosing the best value of k. Figure 12(b), shows the fraction of time (in percentage) inter-site links remain fallow as k varies. We see severe underutilization of resources at $k = 1$, explaining the poor performance of SJF. At peak ($k = 57$), we see that the links are not utilized only 5% of the time. Any further increase in k results in decreased link fallow time; however, higher values of k lead to cases where the initial stages (mappers) of k QEPs get scheduled first, as they are available immediately. As a result, resources are blocked for later stages for all k QEPs, resulting in an increase in average run times.



(a) Performance across queries binned by total amount of intermediate data

(b) Performance across queries binned by total input size of tables

(c) Performance across queries binned by bandwidth skew

Figure 14: Isolating gains observed across queries

C: CLARINET CO: CLARINET-O

	FACEBOOK		MICROSOFT	
	C	CO	C	CO
25%ile	27	13	9	8
Mean	59	30	63	34
75%ile	68	36	67	40
90%ile	72	47	78	48

Table 2: CLARINET vs. a variation allowing overlap of network transfers. HIVE+ is used as the baseline.

Fairness across queries: Recall from §5.3 that CLARINET uses a knob ϵ to ensure fairness: $\epsilon \rightarrow 0$ tends to bias CLARINET's core heuristic (§5.2) to schedule from jobs that are nearing deadlines computed based on their fair share (hence leading to greater fairness), whereas $\epsilon \rightarrow 1$ favors performance at the expense of jobs being delayed beyond their fair-share deadlines. Performance improvements from CLARINET relative to HIVE+ as a function of ϵ are shown in Figure 13(a). We see that even when biasing toward fairness ($\epsilon \rightarrow 0$), CLARINET offers substantial improvements (20%) relative to HIVE+. As we trade-off some amount of fairness (higher ϵ), CLARINET's benefits improve almost linearly. Figure 13(b) shows the percentage of jobs that did not meet the fair deadline as a function of ϵ. For low values of $\epsilon (= 0.1)$, we see that almost 90% of jobs meet their deadline and are not starved by jobs arriving in the future.

Non-overlap: We compare CLARINET with a system, CLARINET-O that disregards CLARINET's schedule and allows tasks to be scheduled as and when they are available. Competing flows on a WAN link will now share the bandwidth equally in space rather than sharing them across time. The QEPs chosen and the placement of tasks are identical in both the cases. Table 2 reports the run time reduction with respect to HIVE+. We note that even with an overlapped schedule, the gains of CLARINET-O over HIVE+ are significant; average run time reduces by 34% (1.5×). This is due to good QEP selection and task placement. Further, CLARINET is 29 percentage points better than CLARINET-O by virtue of combining QEP selection and task placement with non-overlapped transfer scheduling. Overlap results in lower allocation of bandwidth for all contending flows, thereby increasing all queries' completion times.

7.4 Profiling gains of queries (simulation)

To isolate characteristics of queries that contribute to higher performance, we categorize them based on the amount of skew in (i) the intermediate data generated from different stages, (ii) the spread of input data across sites, and (iii) the average outgoing bandwidth of sites where input tables of a query are located. For each characteristic, we split the queries into "bins" based on the normalized standard deviation (COV). Figure 14 presents the performance gains for queries in each bin.

Queries with high skew (> 2) in the amount of intermediate data perform 3× better than queries for which the intermediate data is equally distributed. The absolute improvement over HIVE+ is as high as 82%. A similar trend is observed for queries categorized by the skew in input data. For queries with low skew in intermediate data/input data, all join orders (all possible QEPs) will exercise all links in the topology. Thus, choosing one over another will not offer substantial improvement in performance.

We also observe performance gains improving (48 to 71%) with growing bandwidth skew, but the effect is less pronounced. This is consistent with the high gains observed in a homogeneous WAN substrate (§7.2). CLARINET performance is not intrinsically tied to the presence of high WAN skew.

8 Related Work

We discuss related work on query optimization in §2.2. CLARINET adds to the rich literature on query optimization in both (distributed) database systems [34, 14, 25, 12, 32, 41, 38, 46, 49] and big data analytics stacks [2, 10]. In particular, it shows how to bring WAN awareness into query optimization in a principled fashion.

Other recent work have explored low-layer optimizations to improve GDA query performance. Iridium [35] develops WAN-aware input data and task placement for two-stage MapReduce jobs. Geode [43] develops input data movement and join algorithm

selection strategies to minimize WAN bandwidth usage. Finally, Jetstream [36] proposes using adaptive filtering and local aggregation of data to improve latency. SWAG [21] coordinates compute task scheduling across DCs.

Many of these apply to simple 1- or 2-stage queries [35, 21, 43], whereas CLARINET considers general DAGs. Some also require detailed modifications to existing analytics frameworks [36], whereas CLARINET's design is such that it can be integrated with ease. More importantly, CLARINET operates at a higher layer than all prior systems, by optimizing query plan generation. Thus, CLARINET has a more fundamental impact on query performance. Also, CLARINET is complementary to these prior systems (e.g., [21, 36]).

9 Discussion

Experimental results highlight CLARINET's performance achieved through WAN-aware QEP selection, combined with operator placement and scheduling aspects of the execution framework. While this motivates the need to explore non-traditional query optimization approaches for the geo-distributed settings, there are a few other aspects to consider.

First, the efficacy of CLARINET depends on the availability of known, non-fluctuating bandwidth between DCs. Most software-defined WAN managers provide this abstraction under normal operating conditions. Further, our experiments on Amazon EC2 showed that minor fluctuations in available bandwidth do not adversely affect CLARINET's performance. However, under catastrophic network failures, the bandwidth availability between DCs can change drastically. CLARINET does not have any mechanism to react under such scenarios. Prior works [11, 29, 9] have presented approaches to dynamically change query execution plans under system changes and cardinality estimation errors. Developing similar techniques to adapt CLARINET's execution plan under bandwidth changes is part of our future work.

Second, CLARINET does not leverage performance gains obtained from using techniques that minimize the overall data transferred over the WAN. These include: (i) using bloom-filters to implement joins as semi-joins, and (ii) caching (intermediate) results data from prior queries [43]. While reducing WAN traffic improves query completion time in the geo-distributed setting, the total data sent over the WAN (e.g., determined by the number of common keys in a bloom-filter semi-join implementation) can be large depending upon the dataset. Under such cases, network-aware QEP selection and scheduling of transfers can further reduce aggregate run times even if WAN traffic reduction methods are used.

10 Conclusion

In this paper, we consider the problem of running analytics queries over data gathered and stored at multiple sites inter-connected by heterogeneous WAN links. We argue that, in order to optimize query completion times, it is crucial for the query plan to be made WAN-aware, for query planning to be done jointly with selecting the placements and schedule for the query's tasks, and for multiple queries to be jointly optimized. We design CLARINET, a novel WAN-aware QO that incorporates a variety of novel heuristics for these issues. We implement CLARINET such that it can be easily integrated into existing data analytics frameworks with minimal modifications. Our experiments using an EC2 deployment and large scale simulations show that CLARINET reduces query completion times by 2× compared to using state-of-the-art WAN-aware placement and scheduling. We also show how our scheme can ensure fair treatment of queries.

Acknowledgments

We thank the anonymous reviewers and our shepherd Amol Deshpande for their insightful comments. Raajay and Aditya are supported by the Wisconsin Institute on Software-defined Datacenters of Madison and grants from Google and National Science Foundation (CNS-1302041, CNS-1330308, CNS-1345249).

References

[1] Amazon datacenter locations. https://aws.amazon.com/about-aws/global-infrastructure/.

[2] Apache Calcite - a dynamic data management framework. http://calcite.incubator.apache.org. Accessed 04-27-2015.

[3] Apache Hive. http://hive.apache.org.

[4] Apache Tez. http://tez.apache.org.

[5] Google datacenter locations. http://www.google.com/about/datacenters/inside/locations/.

[6] Microsoft datacenters. http://www.microsoft.com/en-us/server-cloud/cloud-os/global-datacenters.aspx.

[7] Spark SQL. https://spark.apache.org/sql.

[8] TPC Benchmark DS (TPC-DS). http://www.tpc.org/tpcds.

[9] AGARWAL, S., KANDULA, S., BRUNO, N., WU, M.-C., STOICA, I., AND ZHOU, J. Reoptimizing data parallel computing. In *NSDI* (2012).

[10] ARMBRUST, M., XIN, R. S., LIAN, C., HUAI, Y., LIU, D., BRADLEY, J. K., MENG, X., KAFTAN, T., FRANKLIN, M. J., GHODSI, A., AND ZAHARIA, M. Spark SQL: Relational data processing in Spark. In *SIGMOD* (2015).

[11] AVNUR, R., AND HELLERSTEIN, J. M. Eddies: Continuously adaptive query processing. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA, 2000), SIGMOD '00, ACM, pp. 261–272.

[12] BERNSTEIN, P. A., AND CHIU, D.-M. W. Using semi-joins to solve relational queries. *Journal of the ACM 28*, 1 (1981), 25–40.

[13] CALDER, M., FAN, X., HU, Z., KATZ-BASSETT, E., HEIDEMANN, J., AND GOVINDAN, R. Mapping the expansion of Google's serving infrastructure. In *IMC* (2013).

[14] DEWITT, D. J., GHANDEHARIZADEH, S., SCHNEIDER, D., BRICKER, A., HSIAO, H.-I., RASMUSSEN, R., ET AL. The Gamma database machine project. *IEEE Transactions on Knowledge and Data Engineering 2*, 1 (1990), 44–62.

[15] GANJAM, A., SIDDIQUI, F., ZHAN, J., LIU, X., STOICA, I., JIANG, J., SEKAR, V., AND ZHANG, H. C3: Internet-scale control plane for video quality optimization. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)* (Oakland, CA, May 2015), USENIX Association, pp. 131–144.

[16] GRAEFE, G. Volcano: An extensible and parallel query evaluation system. *IEEE Trans. on Knowl. and Data Eng. 6*, 1 (Feb. 1994), 120–135.

[17] GRAEFE, G. The cascades framework for query optimization. *Data Engineering Bulletin 18* (1995).

[18] GRANDL, R., ANANTHANARAYANAN, G., KANDULA, S., RAO, S., AND AKELLA, A. Multi-resource packing for cluster schedulers. In *SIGCOMM* (2014).

[19] GUPTA, A., SUDARSHAN, S., AND VISHWANATHAN, S. Query scheduling in multi query optimization. In *Database Engineering and Applications, 2001 International Symposium on.* (2001), IEEE, pp. 11–19.

[20] HONG, C.-Y., KANDULA, S., MAHAJAN, R., ZHANG, M., GILL, V., NANDURI, M., AND WATTENHOFER, R. Achieving high utilization with software-driven WAN. In *SIGCOMM* (2013).

[21] HUNG, C.-C., GOLUBCHIK, L., AND YU, M. Scheduling jobs across geo-distributed datacenters. In *SoCC* (2015).

[22] ISARD, M., PRABHAKARAN, V., CURREY, J., WIEDER, U., TALWAR, K., AND GOLDBERG, A. Quincy: Fair scheduling for distributed computing clusters. In *SOSP* (2009).

[23] JAIN, S., KUMAR, A., MANDAL, S., ONG, J., POUTIEVSKI, L., SINGH, A., VENKATA, S., WANDERER, J., ZHOU, J., ZHU, M., ZOLLA, J., HÖLZLE, U., STUART, S., AND VAHDAT, A. B4: Experience with a globally-deployed software defined WAN. In *SIGCOMM* (2013).

[24] JIANG, J., DAS, R., ANANTHANARAYANAN, G., CHOU, P., PADMANABHAN, V., SEKAR, V., DOMINIQUE, E., GOLISZEWSKI, M., KUKOLECA, D., VAFIN, R., AND ZHANG, H. Via: Improving internet telephony call quality using predictive relay selection. In *SIGCOMM* (2015).

[25] KITSUREGAWA, M., TANAKA, H., AND MOTO-OKA, T. Application of hash to data base machine and its architecture. *New Generation Computing 1*, 1 (1983), 63–74.

[26] KUMAR, A., JAIN, S., NAIK, U., RAGHURAMAN, A., KASINADHUNI, N., ZERMENO, E. C., GUNN, C. S., AI, J., CARLIN, B., AMARANDEI-STAVILA, M., ROBIN, M., SIGANPORIA, A., STUART, S., AND VAHDAT, A. BwE: Flexible, hierarchical bandwidth allocation for WAN distributed computing. In *SIGCOMM* (2015).

[27] LI, H., GHODSI, A., ZAHARIA, M., SHENKER, S., AND STOICA, I. Tachyon: Reliable, memory speed storage for cluster computing frameworks. In *Proceedings of the ACM Symposium on Cloud Computing* (New York, NY, USA, 2014), SOCC '14, ACM, pp. 6:1–6:15.

[28] MACKERT, L. F., AND LOHMAN, G. M. R* optimizer validation and performance evaluation for distributed queries. In *PVLDB* (1986).

[29] MARKL, V., RAMAN, V., SIMMEN, D., LOHMAN, G., PIRAHESH, H., AND CILIMDZIC, M. Robust query processing through progressive optimization. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA, 2004), SIGMOD '04, ACM, pp. 659–670.

[30] MASTROLILLI, M., AND SVENSSON, O. (acyclic) job shops are hard to approximate. In *FOCS* (2008).

[31] MONALDO, M., AND OLA, S. Improved bounds for flow shop scheduling. In *ICALP* (2009).

[32] MULLIN, J. K. Optimal semijoins for distributed database systems. *IEEE Transactions on Software Engineering 16*, 5 (1990), 558–560.

[33] OLSTON, C., REED, B., SRIVASTAVA, U., KUMAR, R., AND TOMKINS, A. Pig latin: A not-so-foreign language for data processing. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA, 2008), SIGMOD '08, ACM, pp. 1099–1110.

[34] POLYCHRONIOU, O., SEN, R., AND ROSS, K. A. Track join: distributed joins with minimal network traffic. In *SIGMOD* (2014).

[35] PU, Q., ANANTHANARAYANAN, G., BODIK, P., KANDULA, S., AKELLA, A., BAHL, V., AND STOICA, I. Low latency geo-distributed data analytics. In *SIGCOMM* (2015).

[36] RABKIN, A., ARYE, M., SEN, S., PAI, V. S., AND FREEDMAN, M. J. Aggregation and degradation in JetStream: Streaming analytics in the wide area. In *NSDI* (2014).

[37] REN, X., ANANTHANARAYANAN, G., WIERMAN, A., AND YU, M. Hopper: Decentralized speculation-aware cluster scheduling at scale. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication* (New York, NY, USA, 2015), SIGCOMM '15, ACM, pp. 379–392.

[38] RODIGER, W., MUHLBAUER, T., UNTERBRUNNER, P., REISER, A., KEMPER, A., AND NEUMANN, T. Locality-sensitive operators for parallel main-memory database clusters. In *ICDE* (2014).

[39] ROY, P., SESHADRI, S., SUDARSHAN, S., AND BHOBE, S. Efficient and extensible algorithms for multi query optimization. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA, 2000), SIGMOD '00, ACM, pp. 249–260.

[40] SELLIS, T. K. Multiple-query optimization. *ACM Trans. Database Syst. 13*, 1 (Mar. 1988), 23–52.

[41] URHAN, T., AND FRANKLIN, M. J. XJoin: A reactively-scheduled pipelined join operator. *IEEE Data Engineering Bulletin* (2000), 27–33.

[42] VISWANATHAN, R., ANANTHANARAYANAN, G., AND AKELLA, A. Clarinet: Wan-aware optimization for analytics queries. Tech. Rep. TR1841, University of Wisconsin-Madison, 2016.

[43] VULIMIRI, A., CURINO, C., GODFREY, B., PADHYE, J., AND VARGHESE, G. Global analytics in the face of bandwidth and regulatory constraints. In *NSDI* (2015).

[44] WANG, X., OLSTON, C., SARMA, A. D., AND BURNS, R. Coscan: Cooperative scan sharing in the cloud. In *Proceedings of the 2Nd ACM Symposium on Cloud Computing* (2011), SOCC '11.

[45] XIAO, X., HANNAN, A., BAILEY, B., AND NI, L. M. Traffic engineering with mpls in the internet. *Network, IEEE 14*, 2 (2000), 28–33.

[46] XIN, R. S., ROSEN, J., ZAHARIA, M., FRANKLIN, M. J., SHENKER, S., AND STOICA, I. Shark: SQL and rich analytics at scale. In *SIGMOD* (2013).

[47] XIONG, P., HACIGUMUS, H., AND NAUGHTON, J. F. A software-defined networking based approach for performance management of analytical queries on distributed data stores. In *SIGMOD* (2014).

[48] ZAHARIA, M., CHOWDHURY, M., DAS, T., DAVE, A., MA, J., MCCAULEY, M., FRANKLIN, M., SHENKER, S., AND STOICA, I. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *NSDI* (2012).

[49] ZAMANIAN, E., BINNIG, C., AND SALAMA, A. Locality-aware partitioning in parallel database systems. In *SIGMOD* (2015).