CellIQ: Real-Time Cellular Network Analytics at Scale

Anand Padmanabha Iyer[⋆†], Li Erran Li[†], Ion Stoica[⋆]
[⋆]University of California, Berkeley [†]Bell Labs, Alcatel-Lucent

Abstract

We present CellIQ, a real-time cellular network analytics system that supports rich and sophisticated analysis tasks. CellIQ is motivated by the lack of support for real-time analytics or advanced tasks such as spatio-temporal traffic hotspots and handoff sequences with performance problems in state-of-the-art systems, and the interest in such tasks by network operators. CellIQ represents cellular network data as a stream of domain specific graphs, each from a batch of data. Leveraging domain specific characteristics—the spatial and temporal locality of cellular network data—CellIQ presents a number of optimizations including geo-partitioning of input data, radius-based message broadcast, and incremental graph updates to support efficient analysis. Using data from a live cellular network and representative analytic tasks, we demonstrate that CellIQ enables fast and efficient cellular network analytics—compared to an implementation without cellular specific operators, CellIQ is 2× to 5× faster.

1 Introduction

Cellular networks have become an integral part of our digital life in an increasingly mobile connected world driven by the wide adoption of smartphones and tablets. These networks must be designed, operated and maintained efficiently in order to ensure satisfactory end-user experience. To achieve this goal, cellular network operators collect unprecedented volume of information about the network and user traffic. Analysis of this data can provide crucial insights in a number of tasks ranging from network planning (e.g., deployment of new base stations) to network operation (e.g., improving utilization of limited radio resources by interference coordination). The analysis is not limited to network operations—for instance, it can also help cities plan smarter road networks, businesses reach more potential customers, and health officials track diseases [29]. Thus, timely and efficient analysis of cellular network data can be beneficial in a variety of scenarios.

Current state-of-the-art cellular analytics systems continuously collect per connection information such as radio resource usage, associated base stations and handoffs at network elements such as the Mobility Management Entity (MME) and probes deployed in strategic locations. The collected information is then backhauled to centralized servers in batches and ingested into the analysis engine [3, 5, 15, 35]. Such analytics systems are either based on streaming database technology [15] or Hadoop batch processing [5, 35]. Thousands of predefined reports are generated from the data periodically and made available in a dashboard for the experts to view. While these reports provide useful information, we have learned from network operators that they can benefit from *timely* and more *sophisticated* analyses. For example, advanced analytics such as detecting and monitoring spatio-temporal hotspots and tracking popular handoff sequences with abnormal failure rate would enable quick resolution of performance problems.

In this paper, we propose CellIQ, a system for cellular network analytics that builds on top of existing big data cluster computing frameworks. By leveraging domain specific knowledge, CellIQ enables fast and efficient cellular network analysis at scale. The key insight in CellIQ is the observation that cellular network data is naturally represented as a time-evolving graph. In this graph, nodes are network entities such as base stations and User Equipments (UE). Edges represent adjacency of base stations or connections between base stations and UEs.

Stream processing and graph processing has been topics of tremendous interest recently, and hence a large number of proposals exist in both areas. Existing streaming systems such as TimeStream [31] and Spark Streaming [37] do not support streaming graph processing. On the other hand, existing graph parallel systems such as GraphLab [25], PowerGraph [19], GraphX [18] and GraphLINQ [28] are not optimized for operations spanning multiple graphs such as persistent connected components over sliding windows. There are a couple of noticeable exceptions: Kineograph [11] and Chronos [21] focus on constructing incremental snapshots of evolving graphs and optimizing data layout and job scheduling. Differential Dataflow [27] supports incremental computation of algorithms on evolving graphs in the Naiad [30] framework. These systems do not present specific optimizations for cellular network analytics.

In contrast, CellIQ is optimized for cellular network analytics. It leverages domain specific characteristics of cellular networks—its *spatial and temporal locality*—to achieve efficient analysis. CellIQ encodes network specific properties in a time-evolving graph. Connection records per time window are edge properties between UEs and base stations. Aggregate statistics per time window such as radio resource allocation, modulation and

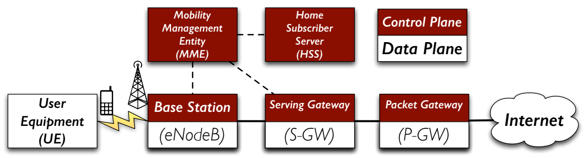


Figure 1: LTE network architecture (description in Table 1).

traffic volume are node properties. It then optimizes the data layout with the use of space-filling curve based geo-partitioning and edge indexing mechanisms. Window operations are efficiently implemented using differential and incremental graph update techniques. To avoid hop-by-hop message propagation, CellIQ enables nodes to broadcast messages to all nodes within a radius. For spatial operations, we further support efficient aggregations.

Using real cellular network data and representative analytics tasks such as spatial and temporal traffic hotspots and popular handoffs, we demonstrate that CellIQ enables real time cellular network analytics. The performance gain can be significant when compared with solutions without cellular specific optimizations.

This paper makes the following contributions:

- We have designed and developed CellIQ, which to the best of our knowledge is the first real-time cellular network analytics system that is capable of running sophisticated tasks including detection and tracking of spatial and temporal traffic hotspots, and popular handoff sequences with abnormal failures.
- We systematically take advantage of the domain specific characteristics of cellular networks, its spatial and temporal locality, to optimize the performance of CellIQ. We succinctly represent a batch of cellular network data as a spatial graph, and continuously arriving data as a stream of spatial graphs. We carefully place data using a geo-partitioning technique that avoids expensive data movements. We propose differential and incremental graph updates for efficient window operations and radius based message broadcast for quicker spatial analysis.
- We evaluate our system using real cellular network data consisting of several thousand base stations and millions of users. Our results show that CellIQ outperforms implementations without cellular specific optimizations vastly.

2 Background on LTE Networks

In this section, we briefly review the LTE network architecture and its data collection mechanism to familiarize the reader with the basic entities in the network and the characteristics of the data available for analysis. We also discuss how existing state-of-the-art analytic systems utilize such collected data.

2.1 LTE Network Architecture

LTE networks enable User Equipments (UEs) such as smartphones to access the Internet. The LTE network architecture is shown in Figure 1, which consists of several network entities (a description is given in Table 1). When a UE is in idle mode, it does not have an active connection to the network. To communicate with the Internet, a UE requests the network to establish a communication channel between itself and the Packet Data Network Gateway (P-GW). This involves message exchanges between the UE and the Mobility Management Entity (MME). The MME may contact the Home Subscriber Server (HSS) to obtain UE capability and credentials. To enable the communication between the UE and MME, a radio connection called radio bearer between the UE and the base station is established. GPRS Tunneling Protocol (GTP) tunnels are established between the base station and the Serving Gateway (S-GW), and between the S-GW and the P-GW through message exchanges involving these entities and the MME. The radio bearer and the two GTP tunnels make up the the communication channel between the UE and the P-GW called Evolved Packet System (EPS) bearer (or simply bearer in short).

When an active UE moves across a base station boundary, its connections will be handed off to the new base station. There are several different types of handoffs: handoffs that require the bearer to be handled by a new S-GW, a new MME, handoffs that require the change of radio frequency or radio technology (e.g. from LTE to 3G). Some of these procedures are very involved. For an active UE, the network knows its current associated base station. For an idle UE, the network knows its current tracking area. A tracking area is a set of base stations that are geographically nearby.

S-GWs are mainly used as mobility anchors to provide seamless mobility. P-GW centralizes most network functions like content filter, firewalls, lawful intercepts, etc. P-GWs sit at the boundary of the cellular networks and the Internet. A typical LTE network can cover a very large geographic area (even as large as a country) and can have a pool of MMEs, S-GWs and P-GWs for reliability and load balancing purposes.

2.2 Data Collection and Analysis

Cellular network operators collect a wide variety of data from their network, a few of which are discussed below:

Bearer and Signaling Records: A UE communicates with the network by establishing one or more bearers. Each bearer may have a different QoS profile or connect to a different IP network. Multiple TCP connections can be carried in one bearer. LTE networks keep track of a rich set of bearer statistics such as (1) traffic volume, frame loss rate in the data link layer, (2) physical radio resources allocated, radio channel quality, modulation and

LTE Architecture Entities	
Name	*Description*
UE	User Equipment: Any device that accesses the network such as smartphones and tablets.
eNodeB	Enhanced Node B: The base station through which UEs access the network.
MME	Mobility Management Entity: Provides roaming and handoff support, UE authentication and paging.
HSS	Home Subscriber Server: Is a central database that contains user and subscription-related information.
S-GW	Serving Gateway: Acts as a mobility anchor.
P-GW	Packet Data Network Gateway: Allocates IP address and centralizes most network functions such as content filter, policy enforcement, lawful intercepts and charging support.

Table 1: Key entities in the LTE network architecture.

coding rate in the physical layer, (3) bearer setup delay, failure reason, (4) associated base station, S-GW, P-GW, MME, and (5) bearer start and end time. LTE networks also collect data on many signaling procedures such as handoff, paging (waking up a UE to receive incoming traffic), attach request. The collection of these data occur at MMEs and base stations, which organize them into records. Each record can have *several hundred* fields. As indicated earlier, LTE networks may have a pool of MMEs, S-GWs and P-GWs. Since a base station can communicate with multiple MMEs, bearer level records need to be merged across MMEs.

TCP Flow Records: Probes can be strategically deployed in the network, e.g., between S-GWs and P-GWs. The purpose of these probes is to collect TCP flow records. The collected flows can then be associated with their corresponding bearer records.

Network Element Records: Network elements such as base stations and MMEs have operational statistics such as aggregate downlink frame transmitted per time window and number of bearers failed per time window. These records are also collected and are normally used for network monitoring purposes.

While collecting data packets continuously is infeasible due to its prohibitive space and resource overhead[1], most of the data mentioned above can be collected without noticeable overhead in operational LTE networks.

Existing state-of-the-art cellular analytics systems are deployed in operator owned data centers. The records that are collected at the network entities are accumulated over short time intervals (e.g., per minute) to be sent to the data center. Although the data is available at minute granularity, existing analysis frameworks do not utilize them as soon as they arrive. Instead, the data is accumulated and used to generate several thousands of pre-defined reports (most of which are aggregate statistics) periodically (typically once a day). The generated reports are displayed in a dashboard where domain experts can peruse them.

[1] An operator may choose to enable packet collection for a short duration for troubleshooting purposes, but such cases are typically rare.

3 Motivation and Overview

Current cellular network analytics systems provide invaluable insights to the network operator. The reports they generate are immensely useful for the operator to understand the behavior of their network. However, in the present form, the analysis supported by these systems are rudimentary at best. Most, if not all, of the generated reports are simple aggregate statistics such as downlink or uplink volume per network entity. We have learned from network operators that cellular networks could benefit tremendously from sophisticated analytics. For instance, operators are interested in learning if some regions of the network are hotspots, and if they are, whether they persist. Since such hotspots may indicate insufficient network resources, they are useful for dynamic load balancing or network planning. Similarly, it is important to detect and mitigate abnormal failures to provide satisfactory end-user experience. Thus, there is a need for cellular analytic systems to be both *timely* and *sophisticated*.

We now outline the challenges in developing such a system and make a case for using cellular specific optimizations to achieve the desired goals. Then we present our solution briefly.

3.1 Requirements and Challenges

An operational LTE network serving a large region can have thousands of base stations and millions of users. To keep up with the demand, operators are continuously adding capacity by deploying new base stations. The number of other network elements, such as MMEs, S-GWs, and P-GWs are also on the rise. A typical LTE network can generate several terabytes of monitoring data per minute. The volume of monitoring data has been growing as both mobile data-plane traffic and signaling traffic (known as the signaling storm problem due to chatty applications) continue to grow exponentially. Due to the interplay between the network elements, LTE network data has to be analyzed as a whole.

Cellular network operators need to perform a *myriad* of analytics tasks in real time. For example, Motive [4] provides over 7000 offline network analytics reports. Performing these and more advanced analyses real-time means that each computationally intensive task must be executed efficiently to avoid impacting the overall system. To illustrate the challenges involved in performing these analysis, we present three broad tasks that are of interest to network operators:

Continuous monitoring of connections and entities: Operators must continuously monitor millions of UEs, their connections and network elements. Fine-grained location and time-dependent thresholds are needed to prevent unacceptable error alarms.

Real time detection of spatial and temporal patterns: Cellular networks exhibit rich dynamics in both temporal and spatial domains. User perceived performance tends to vary over time and location due to changes in the subscribers' activity. Hence, operators need to detect and track spatial and temporal patterns. Examples of such patterns include (1) persistent spatial hotspots in terms of abnormally high signaling traffic, or high frame loss rate, and (2) impending flash crowd events that draws a large number of users to the same location.

Real time troubleshooting to identify root causes: Operators need to perform sophisticated on demand analytics tasks to understand the root cause of performance and security problems. Similar to wired networks, cellular network operators need to detect, locate and troubleshoot performance and security problems, e.g., via expert rule-based inference [24], machine-learning techniques [1, 13], or inference of dependency among network elements, entities and events [7, 22]. Performing these tasks in real-time can be very challenging.

3.2 Need for Cellular Specific Optimizations

Having discussed the various requirements and challenges, we now turn our attention towards how a data processing system may accommodate such analysis tasks. To do so, we contacted network administrators and discussed a few representative analysis tasks of interest. We then implemented these tasks in an existing graph-parallel analysis framework. During this exercise, we realized that most, if not all, of the analysis tasks can be expressed by three operations: (i) sliding window operation, (ii) time window operation[2], and (iii) spatial operation. This section is a reflection of our experience, describing how these operations can be used, as available in existing frameworks, to implement a typical analysis task. In each of the example tasks, we detail why a straight-forward implementation may not be sufficient, alluding to the need for domain specific optimizations.

3.2.1 Sliding Window Operations

Persistent hotspot tracking per sliding window We define a traffic hotspot to be a group of close by base stations, each of whose traffic volume is above a threshold for the time window (referred to as snapshot). We can construct a graph with nodes representing base stations. Two nodes are connected by an edge if and only if the traffic volumes of both nodes exceed the threshold. Detecting hotspots is then equivalent to computing connected components on this graph. Although the computation can be expressed in a distributed dataflow framework [30, 36] using `join` and `group-by` operators, this can be very inefficient as shown in prior work in graph parallel systems [18, 25, 26]. The reason is that join operators are not optimized for graph processing and can be very expensive. Network operators need to compute traffic hotspots per time window. In addition, they also need to detect persistent hotspots (a hotspot is persistent for a sliding window if it is a hotspot in all the component intervals). This task thus requires computation across many windows.

As a concrete example, Figure 2a shows the hotspot graph for three time windows. Suppose we want to compute the persistent hotspots for a sliding window of 3. A straightforward approach is to merge the graphs and perform connected component computation on the resulting graph using a graph parallel processing engine such as GraphLab, GraphX, or GraphLINQ [18, 25, 28]. However, we found that this strategy to be very inefficient when the sliding window is large.

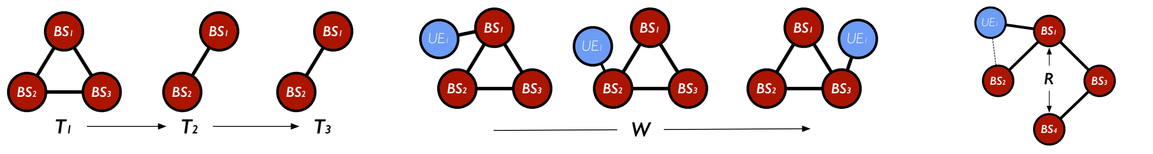
A better approach is to maintain a cumulative graph which counts the number of edges. We then subtract the edge counts from the time window that needs to be forgotten, thus applying *differential* updates to the underlying graph. For the example presented earlier, the edge count for BS1—BS2 is 3, while that for BS2—BS3 and BS1—BS3 is 1. Suppose the graph at time window 0 is empty. We can perform the computation on this cumulative graph. Two nodes are in the same connected component iff their edge count is 3. Hence, the persistent hotspot is BS1—BS2. We demonstrate that this technique speeds up sliding window operations by up to 3× (§ 6).

3.2.2 Time Window Operations

Popular handoff sequence tracking per time window Handoffs can cause connection failures or performance degradation. Operators are interested in monitoring popular handoff sequences in time windows and across sliding windows. A handoff sequence is a valid base station traversal sequence by a set of UEs. If we keep track of both the sequence and the set of associated UEs, then handoff sequence tracking can be implemented as an iterative graph algorithm. Consider the example in Figure 2b, where UE1 is handed off from base station BS1 to BS2 and then from BS2 to BS3 over a time window W. If we are interested in computing the popular handoff sequences in this window W, then BS1 sends the ID of UE1 and the sequence it observes, BS1→BS2, to BS2. In the next iteration, BS2 appends the sequence with its observation, and forwards the new sequence, BS1→BS2→BS3, to BS3. A shortcoming of this approach is that the state management and iterations required to converge becomes a bottleneck for large windows. Thus, the analysis slows down significantly.

A simple optimization to this strategy is to divide W into smaller windows w. However, we cannot compute sequences in each of these windows independently and

[2]It may appear that (i) and (ii) are similar, but it is important to note the difference. Sliding window operations requires more state management, since records in one window may be reused in the next. In other words, the *expiry* of records in a sliding window are variable, while that in a time window are same.



(a) Monitoring persistent hotspots. **(b)** Popular handoffs in a window. **(c)** Traffic gradients.

Figure 2: Representative analysis tasks of interest to cellular network operators.

then combine them[3]. Instead, we bootstrap every window w with the previous window's handoff sequence and *incrementally* update the graph. Applying this technique results in speeding up analysis tasks that depend on time window operations by 2× to 5× (§ 6).

3.2.3 Spatial Operations

Top traffic gradients tracking Users may converge to a particular location. Operators need to predict these movements and re-optimize their network (e.g., self-optimization techniques such as antenna tilt adjustment and interference coordination) to handle such situations. A traffic gradient of a base station is defined as the weighted average of traffic moving towards it. A coarse grained approximation is to consider all handoffs around a distance of a certain radius R. For each handoff, we project the speed towards the base station and weigh by the product of the bearer throughput divided by the distance between the source base station and the base station under consideration. For example, in Figure 2c, suppose there is a handoff of UE1 from BS1 to BS2. The handoff information will propagate to all base stations in a radius R. $BS4$ (not a direct neighbor of either BS1 or BS2) will add the traffic gradient of this handoff to its current gradient. Currently most graph processing systems only propagate message on a hop-by-hop basis, thus this analysis would be inefficient if implemented directly. A better approach is to *broadcast* the message to a multi-hop neighborhood in one iteration. We found this optimization to speed up this analysis by up to 4× (§ 6).

3.3 Solution Overview

The examples we discussed previously show that cellular network analytics systems require a computation model that can process property graph streams efficiently. To achieve this goal, we presented a case for leveraging cellular specific optimizations exploiting spatial and temporal locality. Ideally we would like a single processing engine that can support a combination of incremental data-parallel processing, stream processing and graph-parallel processing. Since the Berkeley Data Analytics Stack (BDAS) [34] supports all of these computation models, we chose to build CellIQ on BDAS. However, we note that the techniques we present are not restricted to a particular framework; for instance, CellIQ's spatial optimization techniques can be incorporated into a different framework such as Naiad [30].

The key abstraction of the BDAS stack is called Resilient Distributed Datasets (RDDs) [36] which can recover data without replication by tracking the lineage graph of operations that were used to build it. GraphX [18], BDAS's graph-parallel engine, builds on top of the RDD abstraction. It represents graph structured data (called property graph) as a pair of vertex and edge property collections (implemented as RDDs). GraphX embeds graph computation within the Spark distributed dataflow frameworks and distill graph computation to a specific join-map-group-by dataflow pattern. It introduces a range of optimizations both in how graphs are encoded as collections and as well as the execution of the common dataflow operators.

CellIQ is implemented as a layer on top of GraphX and incorporates several domain specific optimizations:

- **Data placement** We implement geo-partitioning of the input data. Vertex properties, edge properties and graphs from different snapshots are co-partitioned. This minimizes data movement.
- **Radius based message broadcast** For messages that need to reach a radius of nodes, we enable the exchanges to complete in one iteration.
- **Spatial aggregation** We implement spatial aggregation for tasks that depend on aggregate statistics such as intra-tracking and inter-tracking area handoff monitoring.
- **Differential graph updates** Tasks that require sliding window operations are implemented using differential updates to the underlying graph over the time windows under consideration.
- **Incremental graph updates** Time window operations are optimized using incremental updates.

We wrap these optimizations with a cellular specific programming abstraction, *G-Stream*. The G-Stream API exposes a domain-specific combination of streaming and graph processing. In the rest of the paper, we describe the CellIQ system (§ 4) and the optimizations (§ 5) in detail.

[3]Doing so without extensive state management would entail incorrect results, because computing sequences independently would miss some subsequences that happen across windows.

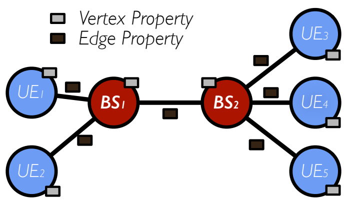


Figure 3: LTE network monitoring data as property graphs.

4 CellIQ System

In this section, we describe how CellIQ represents cellular network data, and optimizes the placement for efficient analysis. We then discuss the computational model.

4.1 Graph Representation

Cellular monitoring data as property graphs Our data model for a window of cellular monitoring data is a graph $G(V,E)$, where the vertex is either a user equipment or a base station. For the purpose of the analytics we are interested in, we discard other entities, although it is easy to incorporate them if required. An edge is formed between a user and the base station to which she is connected, Thus, each base station vertex consists of many edges. Similarly, an edge is formed between two base stations when a user traverses between them (i.e., she is handed-off from the first base station to the second). To access the cellular network, a UE performs various procedures during which it exchanges control messages with the network. These procedures are carried out using complex protocols modeled as state machines. Any action the user wishes to take in the network, such as browsing the web, watching a video or making a voice call, triggers a myriad of control plane messages. We incorporate these control plane monitoring records as edge properties. Handoffs records are edge properties of the previous base stations. We do not replicate the record for the new base stations. Essentially, all control plane interaction between a user and a base station is stored on the edge between them. Similarly, the edge between two base stations may store records relevant to user traversals between them. Figure 3 depicts such a simple graph.

This representation enables us to do computations on the control plane data efficiently. For instance, the path of a user can be found using a simple graph traversal. Similarly, aggregate base station information can be obtained without any costly map and shuffle operations.

4.2 Graph Partitioning

A straight-forward approach to distributing the graph in a cluster is to partition the vertices and edges among the machines using a hash-partitioner. Although such a scheme ensures uniform distribution of vertices and edges, it also places neighboring vertices in different machines, thus resulting in poor performance. PowerGraph [19] introduces the vertex-cut technique based on the observation that natural graphs exhibit power law degree distribution, and proposes a greedy edge placement algorithm that minimizes vertex replication. Cellular network graphs do not typically exhibit power law degree distribution and hence the algorithm is not directly applicable. An alternative approach is to use balanced edge-cuts (e.g., as proposed by METIS [23]) for partitioning. However, this results in two disadvantages. First, they result in edge replication which are costly in our graph representation with many thousands of records stored in edges per time window. Second, since edge properties change over time, an edge-cut that is optimal in one snapshot may not remain so in the next, requiring expensive data movements. To strike a balance between these advantages and shortcomings, CellIQ uses the vertex-cut strategy to avoid replicating edges, and a geo-partitioning technique to place edges to preserve spatial locality.

Geo-partitioning of data For efficient analysis of cellular network data, CellIQ requires nodes that are physically close by to be present in the same partition. To achieve this, CellIQ uses a geo-partitioner to distribute the graph across the cluster. The partitioner first maps the vertices to real world geo-coordinates. Each user inherits the location of the base station they are connected to. A standard way to organize multidimensional co-ordinates is to use a tree based datastructure (e.g., Quad-trees [17] or R-trees [20]), but they require complex look ups in a distributed setting. In order to leverage the key based look up schemes typical in cluster computing frameworks, CellIQ's geo-partitioner uses a space-filling curve based approach. The key idea behind space filling curves is to map 2-dimensional locations to 1-dimensional keys that preserve spatial proximity [33]. Thus, keys that are contiguous represent contiguous locations in space. We convert the geo-location of each of the nodes in the graph to its corresponding 1 dimensional space-filling curve key. The key space is then range-partitioned to the machines in the cluster. Edges are co-partitioned with vertices by assigning them the key associated with the source vertex.

Edge indexing Many base station properties such as traffic volume, aggregate frame losses are computed from edge properties between base stations and UEs. To enable fast computation, we index edge properties by base station ID. This ensures the edge properties of a given base station will most likely end up in one partition.

4.3 Computation Model: Discretized Graph Streams (G-Streams)

Because cellular network data arrives continuously, we need to perform the analysis tasks in a streaming fashion. We treat a streaming computation as a series of deterministic batch graph computations on small time intervals (snapshots). The data received in each interval is stored in the cluster to form an input dataset for that interval. Once the time interval completes, this dataset is processed via deterministic graph parallel operations, such as subgraph, connected component, etc to produce new datasets representing either program outputs or intermediate state.

We define *discretized graph streams (G-Streams)* as a sequence of immutable, partitioned datasets (property graphs as a pair of vertex and edge property collections) that can be acted on by deterministic *transformations*. User defined cellular network analytics programs manipulate G-Stream objects. In contrast, D-Streams are defined on a sequence of RDDs instead of property graphs. We will show that our computations cannot be easily expressed using the D-Stream API.

5 CellIQ API and Optimizations

In this section, we present the APIs and various optimization techniques in CellIQ.

5.1 GeoGraph API

The GeoGraph represents the domain specific property graph presented in the previous section, and incorporates the spatial optimizations in CellIQ. The methods exposed by the API is shown in Listing 1.

```
class GeoGraph[V, E] extends Graph[V, E]{
     ...
  //For efficient message exchanges
  def sendMsg (radius: Double, V, V) : M

  //For spatial aggregation tasks
  def spatialAG(reduceV: (V, V) => V,
                reduceE: (E, E) => E) = {
    val superV: Collection[(ccId, V)] =
        this.vertices.groupBy(ccId, reduceV)
    val superE: Collection[(ccId, ccId, E)] =
        this.triplets.map
            { e => (e.src.cc, e.dst.cc, e.attr) }
           .groupBy((e.src.cc, e.dst.cc), reduceE)

    //Return the final graph
    Graph(superV, superE)
  }
}
```

Listing 1: Spatial graph API for cellular monitoring data.

5.1.1 Message Broadcast Within a Radius

Similar to the traffic gradient tracking example presented earlier, many analysis tasks may require messages from a node in the graph to be propagated to every other node within a geographic distance that far exceeds a single hop[4]. GraphX implements this operation using triplets (a triplet contains an edge and its property, and the two component vertex properties), which requires join operations. Since the operation has to be repeated for every iteration (hop), it becomes expensive. The sendMsg API is designed to enable efficient message broadcast to multiple nodes rather than just the immediate hop neighbor. It uses a routing table similar to the one maintained by GraphX. These routing tables are maintained in the vertex partitions and identifies the edge partitions that have edges associated with each vertex in the vertex partition.

In CellIQ, the edges are defined by a distance threshold. We decompose the entire space of interest into subspaces using the threshold by overlaying a grid. For each node, we can compute the subset of subspaces that may contain nodes within a radius R. We maintain a subspace to edge partition mapping that enables easy lookup. This approach is much more efficient than the hop-by-hop propagation, as it minimizes the overheads of joins to a constant instead of being proportional to the hop count.

5.1.2 Spatial Aggregation

Many classes of analysis require operations on spatially aggregated graphs. For instance, operators are interested in tracking intra-tracking area and inter-tracking area handoffs. A tracking area consists of a set of base stations. Inter-tracking area handoffs are more involved which consume high signaling resources and are thus more prone to failures. To assist this kind of tasks, CellIQ exposes the spatialAG function. The function assumes that each graph vertex contains a field cc that can be used for vertex and edge grouping. The function takes two reduce functions: one for aggregating vertex properties and the other for aggregating edge properties.

As an example, to compute inter-tracking area and intra-tracking area handoffs, we could use the tracking area ID (TAI) field as the cc field. Our vertex reduce function would sum up each component vertex's property fields such as traffic volume. If we are only interested in handoffs, then this function may return null. For the edge reduce function, we return the total handoffs. Note that we allow self-edges. A self-edge property is the sum of intra-tracking area handoffs.

5.2 GStream API

The GStream API in CellIQ is as described in Listing 2. The input to CellIQ is a stream of GeoGraphs. Similar to DStreams [37], we implement operations on this streaming domain specific graph by batching their execution in small time steps. In our system, input graph streams are read from the network. Two types of operations apply to these graph streams: (1) *Transformations* create a new G-Stream from one or more parent streams. These can either be *stateless*, applying separately on the property graph in each time interval or stateful, producing states across time intervals. (2) *Output operations*,

[4]In metropolitans, base stations may be placed as close as a few hundred meters from each other, while the analysis may look at areas spanning several miles.

```
class GStream[V, E] extends Serializable {
    ...
def vertexStream(): DStream[(Id, V)] =
    this.map(g => g.vertices)
def edgeStream(): DStream[(Id, Id, E)] =
    this.map(g => g.edges)

def graphReduce(reduceFunc(Graph[V, E], Graph[V, E],
    fv: (V, V) => V, fe: (E, E) => E)
    ): Graph[V, E] =
    this.reduce((a, b) => reduceFunc(a, b, fv, fe))

// Return a new Gstream by reducing the input graph
// over a sliding window. fv and fe can use defaults
// if not supplied. The differential version (not
// shown) also requires an inverse reduce function.
def graphReduceByWindow(
    reduceFunc(Graph[V, E], Graph[V, E],
               fv: (V, V) => V,
               fe: (E, E) => E): Graph[V, E],
    windowDuration: Duration,
    slideDuration: Duration
    ): GStream[V, E] =
    this.window(windowDuration,
    slideDuration).map(x => x.graphReduce(reduceFunc))
}
```

Listing 2: GStream API.

similar to Spark, write data to external systems.

Even though we can not directly extend D-Stream API, we provide two functions that maximally reuse D-Stream functions. The two functions convert a G-Stream into an independent vertex property D-Stream and edge property D-Stream. These can use all the original D-Stream functions, specifically the functions on collections of key value pairs. The key for the vertex D-Stream is the vertex ID and value is the vertex property. Similarly, the key for the edge D-Stream is the edge ID and value is the edge property. Since the individual component RDDs (vertex or edge RDDs) of the D-Stream are geo-partitioned, they automatically take advantage of our spatial optimizations.

G-Streams support the same stateless transformations available in GraphX including subgraph, connected components and join of vertex and edge RDDs. In addition, G-Streams also provide several *stateful* transformations for computations across multiple time intervals.

Windowing: Similar to D-Stream windowing operator, the *window* operation groups all the graphs from a sliding window of past time intervals into one. For example, calling gs.window("5s") yields a G-Stream containing graphs in intervals [0,5), [1,6), [2,7), etc.

graphReduce: Reduces a G-Stream into a GeoGraph.

Sliding window: The *graphReduceByWindow* operation computes one graph per sliding window.

5.2.1 Extending GraphX Operators to Support `graphReduce`

We represent each time window of data as a property graph. To perform window computations, we need to reduce a sequence of graphs into one graph. GraphX does not support certain graph transformations such as intersection and union. We extend the GraphX API to support these transformations. Both `intersection` and `union` operators take two graphs, a vertex function and an edge function. The vertex function decides what to do with the vertex properties of each common vertex. Similarly, the edge function decides how to combine the edge properties of each common edge. An `intersection` operator performs a GraphX `innerJoin` operation on either two vertex or two edge RDDs, and keeps only common vertices and edges in both graphs. A `union` operator performs a GraphX `outerJoin` operation and keeps all vertices and edges from both graphs.

```
def persistConnectedComponents(gs: GStream) = {
    val gs1 = gs.graphReduceByWindow(
          (a, b) => a.intersection(b,
                    (id, v1, v2) => _,
                    (id1, id2, e1, e2) => _),
          "1s", "5s")
    val hotspots = gs1.map(_.connectedComponents())
}
```

Listing 3: Connected components in sliding windows.

Listing 3 illustrates the use of the `graphReduce` operator by computing the connected components in each sliding window, where we reduce each sliding window into a graph using the `intersection` operator as the reduce function. We then output the connected component in each sliding window of 5s using the `connectedComponents` operator of GraphX. Similarly, to compute popular handoff sequences for each sliding window, we collect all handoff sequences for each sliding window using the `reduceByWindow` operator. We then sort the sequences by the number of UEs traversing them.

5.2.2 Differential Updates for Sliding Window Operations

For sliding window computation, if we have to perform pair-wise graph reduce operation, it can be very expensive. To enable differential computation, we provide differential aggregation of property graphs. The differential version of `graphReduceByWindow` takes an graph aggregation function and a function for "subtracting" a graph. The incremental computation can be implemented in this framework by using a null subtraction function and then resetting the graph at every window.

In the example shown in listing 4, the reduce function simply sums up the vertex properties (counts) and edge properties (counts) of the two graphs. The inverse reduce function just subtracts the vertex properties and edge properties of one graph from the other. For each sliding window of K snapshots, instead of computing $K-1$ graph intersections, we only perform one graph union and one graph subtraction. We union the cumulative graph with the graph of the current snapshot, and subtract the

```
def persistConnectedComponents(gs: GStream) = {
    val gs1 = gs.graphReduceByWindow(
               (a, b) => a.union(b,
                  (id, v1, v2) => v1+v2,
                  (id1, id2, e1, e2) => e1+e2),
               (a, b) => a.intersection(b,
                  (id, v1, v2) => v1-v2,
                  (id1, id2, e1, e2) => e1-e2),
               "1s", "5s")

    val hotspots = gs1.map(x =>
               x.subgraph(vPred = (id, c) => c>=K,
                          ePred = (id1, id2, cV1,
                                cV2, cE) => cE>=K)
                .connectedComponents())
}
```

Listing 4: Incrementally computing connected components in each sliding window.

graph at $t-K$ time interval where t is the current interval number. To compute the persistent hotspots for a sliding window, we filter vertices and edges whose count are smaller than K using the subgraph operator of GraphX, and then run `connectedComponents`.

Similarly, for handoff sequence, we accumulate the list of UEs traversed a handoff sequence (list combine). For subtraction, we just remove the tail elements of the sequence from $t-K$ time interval.

5.3 Co-partitioning Component Graphs

As shown in Chronos [21], in general, it is very hard to accommodate graph structure locality (neighborhood) per snapshot and temporal locality (co-locate vertices or edges in different time windows) across snapshots. Applications or systems have to make a tradeoff between retaining structure locality and temporal locality for evolving graphs. In cellular network data, edges in one snapshot have spatial locality and edges across snapshot retain most of the spatial locality as users do not move long distance over short time windows. As a result, we co-partition all graph snapshots in the active set (old snapshots are cleaned up). This co-partition retains both structural and temporal locality, and significantly reduces data movement for computations on G-Streams.

5.4 Indices and Routing Tables

GraphX maintains indices on the partitions that vertices or edges reside. It also keeps a routing table so that a vertex can find out which edge partitions contain its neighbors. We share the same index and routing data structures for all component graphs in a G-Stream since we co-partition the component graphs.

6 Evaluation

We evaluated CellIQ's performance using the three representative analysis tasks we presented in § 3. Our results are summarized below:

- Geo-partitioning has a significant impact in CellIQ's performance. The improvement due to this partitioning strategy ranges from 2× in small analysis windows to several orders of magnitude in larger windows. In addition, geo-partitioning enables analysis to complete when other partitioning strategies fail due to the data movement overhead.
- CellIQ's incremental graph update strategy results in the reduction of analysis time by 2× to 5×.
- The differential graph update technique significantly benefits sliding window computations, by improving performance by up to 4×. Moreover, the technique enables CellIQ to perform well for various window sizes, when strawman techniques incur increasing performance penalty when the analysis window becomes larger.
- Radius based broadcast improves the analysis time by up to 4× compared to the standard hop-by-hop propagation approach.

We discuss these results in detail in the rest of this section after describing our evaluation set up and the datasets used in our experiments.

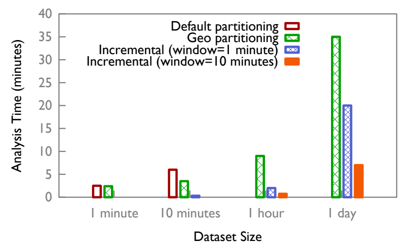
Evaluation Setup: Our evaluation environment consists of 10 machines forming a cluster. Each machine consists of 4 CPUs, 32GB of memory and a 200GB magnetic hard disk. In addition to HDFS, a network storage of 1TB is accessible from all the machines. CellIQ system was built on GraphX version 1.0.

Dataset: We obtained LTE control plane data from a major cellular network operator. The data is from a live network which serves around 1 million subscribers in a large metropolitan area. A single file is generated every minute, and contains around 750,000 records. We receive 10 such files every minute from 10 collection points, bringing the total number of records per minute to approximately 7.5 million. Thus, in the following experiments, we process 450 million records for window sizes of 1 hour and 4.5 billion records[5] for a day window. We store a week worth of data in HDFS, which accounts to approximately 2 terabytes of compressed data.

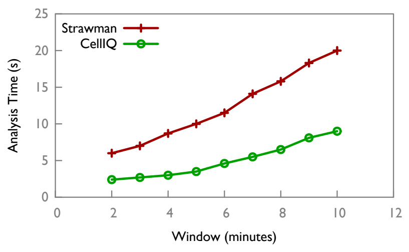
6.1 Tracking Popular Handoff Sequences

With the increase in base station deployment in an effort to combat the increasing demands in data traffic, handoffs become inevitable when users are mobile even to a small extent. While most handoffs are benign, analyzing handoff patterns often helps operators uncover end-user performance issues. For instance, ping-pong handoffs may indicate an incorrect base station configuration, and unexpected handoff sequences seen by many users may indicate interference issues. The results from this application can be combined with other metrics, such as downlink throughput, to uncover problematic sequences.

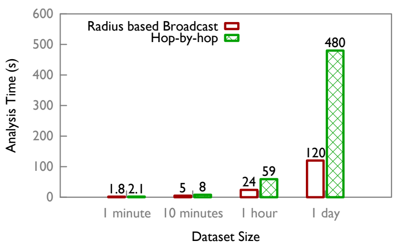
[5]The operator collects data only during the 10 most active hours.



(a) Partitioning and incremental updates.



(b) Differential updates.



(c) Radius based message broadcast.

Figure 4: Partitioning and incremental update has a significant impact on the analysis time (a missing value in 4a indicates either an invalid analysis such as 10 minute incremental window on 1 minute analysis, or a timeout due to memory issues). Sliding window computations benefit from differential updates. Radius-based broadcast can further improve the performance on large datasets.

We implemented this in CellIQ using a program that closely matches Pregel. The program takes in a window W, and outputs the top N sequences in the window. The program bootstraps by assigning each edge information on the users that traversed them along with their count. The vertices (base stations) book-keep the handoff sequences, initially an empty set. At every iteration, the vertices send messages to their neighbors. The message consists of the users who traversed from the source vertex to the destination vertex. Clearly, the bootstrap message sends all users who were present at the source vertex at window start. Subsequent messages consist of users who reached the source vertex from other vertices. Thus, after k^{th} iteration, each vertex learns about a handoff sequence of length $k+1$. The algorithm converges when there are no more messages to send.

Benefits of geo-partitioning: To understand the benefits of data placement, we ran this program on datasets of varying sizes, namely 1 minute, 10 minutes, 1 hour and 1 day, with two partitioning schemes. The default data placement distributes the edges across machines so as to balance the load[6]. In contrast, CellIQ's geo-partitioner uses the location of the source vertex as the key. The results of the comparison of performance of the two schemes are depicted in figure 4a. The gains of data placement are clear from the results, which indicate improvements ranging from 2× (smaller datasets) to several orders of magnitude (larger sets) for the geo-partitioned case[7]. As expected, we see the benefits increase with the size of the dataset. The primary reason for this behavior is the locality achieved by the partitioner. When nodes that are geographically close by are placed in the same partition, the number of messages that a node needs to send to other partitions are reduce drastically. The reduction closely matches the performance difference.

Benefits of incremental graph updates: Next, to evaluate the performance of the incremental graph update technique, we reran the analysis program with a few small changes. Instead of running the program on the entire window W, we break up W into smaller windows w. Rather than naively running the program every w and then combining the results, CellIQ uses the `graphReduce` operations (§ 5) that maintain the result from every window w. The next window is bootstrapped from this result. The analysis time for running this program on the same dataset is shown in figure 4a.

We were surprised to see the benefits of this strategy, the reduction in analysis time showed a factor of 2× to 5×. Upon closer evaluation, these benefits come from two sources. First, the incremental update limits the amount of graph unions performed to one. Second, when the analysis graph is kept smaller, the number of messages to be sent in each iteration is reduced. An interesting observation is that the performance of the incremental strategy is better with 10 minute window compared to 1 minute window. Increasing the window size to 15 results in a lower performance compared to 10 minutes. We tried experimenting with different window sizes, and found that very small windows tend to have poor performance. Due to the lack of space, we do not present the results. Finding the optimal window size that minimizes the analysis time is beyond the scope of this work, and may be obtained using techniques similar to those detailed in [16].

Benefits of differential graph updates: Finally, to evaluate the efficacy of differential updates on sliding windows, we conducted the following experiment. We streamed one day's data to CellIQ. The handoff analysis is done on this data in batches of 1 minute and slide durations of 1 minute. We varied the window of analysis from 2 minutes to 10 minutes. Thus, a window of 2 minutes indicate that every minute (slide duration), the system computes handoff sequences for the last 2 minute data. The strawman approach keeps a ring buffer where it saves the graph every batch. Then, at the slide window, it combines all the graphs and computes handoff sequences. In

[6]We also used the 2D partitioner in GraphX, but results were similar.

[7]In large datasets, the default partitioner failed to run due to the number of messages generated.

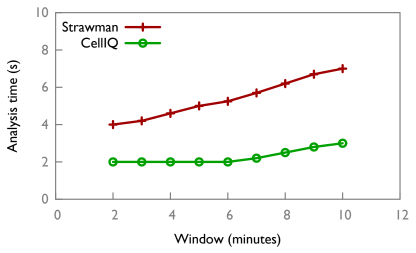


Figure 5: CellIQ's differential update strategy is able to scale simple and complex graph algorithms well.

contrast, CellIQ uses `graphReduceByWindow` that maintains a cumulative graph of the number of users traversing edges. Thus, at every slide window, it only needs to subtract the first graph in the window to obtain the graph on which the analysis needs to be performed. Both approaches use geo-partitioning. The results from this experiment are shown in figure 4b. The strawman approach is able to compute handoffs relatively easily when the window sizes are small. However, when the analysis window is increased, it needs to join many graphs to obtain the result. In comparison, CellIQ is able to scale well due to the fixed number of operations it performs to compute the results. The performance improvement of CellIQ ranged from 2× to 3× in this experiment.

6.2 Monitoring Persistent Hotspots

Arguably, the popular handoff tracking task uses a reasonably complex algorithm that depends on iterative messaging. How does CellIQ's differential update strategy perform on standard graph algorithms that are not message heavy? To answer this question, we implemented an analysis task that continuously monitors hotspots in a given region. As mentioned earlier, hotspot computation can be represented as finding connected components in a graph. Similar to the task before, we use a strawman that builds a graph for every batch and saves it in buffer. At each slide interval, it analyzes all the saved graphs and runs connected components on the union. In contrast, CellIQ leverages its `graphReduceByWindow` operation and then applies connected components on the result.

We again used one day worth of data for this experiment. The hotspot analysis is done on this streaming data in batches of 1 minute and slide durations of 1 minute. We varied the window from 2 minutes to 10 minutes. The average values are presented in figure 5. We find results similar to the popular handoff monitoring task, except that the analysis runs faster because of the lower messaging overhead. Thus, CellIQ's differential update technique benefits all sliding window operations.

6.3 Computing Traffic Gradients

Finally, we evaluate the radius based message broadcast in CellIQ. To do so, we use a task that computes the gradients of base stations in a given interval. As we discussed in § 3, such analysis can be very useful for network operators in the context of optimizing their network. Consider, for instance, a large crowd moving towards an area (e.g., popular events). In these cases, it is desirable to provision additional capacity in the area of gathering. Today, operators need to pre-provision capacity using advance knowledge of the events.

In our program, vertices (base stations) need to send the gradient of their users to their neighbors. The propagation of a message stops when it reaches a neighbor at radius r. While this looks similar to our earlier example of handoff analysis, there is one key difference: the message sent in every iteration is the same. Hence, CellIQ utilizes radius based message broadcast to avoid the penalty associated with multiple iterations. Comparison of this approach against the standard hop-by-hop iterative approach is depicted in figure 4c. The approach performs very well when the input dataset is large, providing gains of up to 4×. Although the number of messages remain same in both approaches, the need to do hop-by-hop propagation impacts the analysis time. Due to low number of messages, smaller datasets can leverage transport layer optimizations that reduce the relative gain. Such optimizations are limited in larger datasets.

7 Discussion

CellIQ leverages domain knowledge to do efficient analysis. A domain focused approach is likely to raise several concerns. We discuss some concerns about CellIQ, focusing on the versatility and generality of its techniques.

How versatile is CellIQ's API?

The representative analyses we present in this paper are the result of our discussions with cellular network operators. Even though our discussion ended with a long list of analysis requirements, we realized that most of them distilled down to one or a combination of the techniques we propose in this work. Thus, we believe that CellIQ is able to accommodate a large set of analysis requirements, and is not restricted to the examples presented here. It is possible that new requirements would need to be accommodated in the future. Since all of our techniques are built on two fundamental datastructures—`GeoGraph` and `GStream`—operators can develop new analysis tasks using these as the building blocks without significant effort.

How general are CellIQ's techniques?

Though CellIQ's primary motivation is to provide timely and efficient cellular network analytics, we believe that the techniques presented in this paper are widely applicable beyond the cellular networks domain. An area of emerging interest is smart-cities, where transportation system optimization is a key challenge. Our graph partitioning (§ 4) and spatial optimization techniques (§ 5) can easily be extended to do traffic analysis on a large scale. Similarly, another domain that has received significant attention recently is the Internet-of-Things (IoT), which also exhibit spatio-temporal characteristics. While our techniques may not carry over to the IoT domain directly[8], we believe that they could be extended to fit the requirements. We envision generalizing the techniques we presented to arbitrary graphs as part of future work.

Can CellIQ be used for real-time feedbacks?

The analysis we discuss in the paper are focused on providing reports—insights and issues in the network—useful for the network operator. A better scenario is to automatically utilize the insights without human intervention. Can CellIQ support such tasks?

Timely processing of data is of prime importance to providing real-time feedbacks. That is, once the data arrives, it is desirable to process it as fast as possible. We designed CellIQ with fast and efficient analysis as key requirements. Such quick analysis enables CellIQ to be useful for providing real-time feedbacks that can be incorporated into the network. Analyzing the efficacy of feedbacks is not within the scope of our work because of the lack of support for configuration change and/or feedback integration in current generation LTE networks. However, with the increasing interest in Self-Organizing Networks (SON), we see this as a venue for future work.

8 Related Work

Cellular network analytics systems Several deployed cellular network analytics system [3, 15] are based on streaming databases. Like other streaming databases such as Aurora, Borealis, STREAM, and Telegraph [6, 8, 10, 12], it is very hard and inefficient to perform iterative graph parallel computations. CellIQ is designed to support real time domain specific streaming graph computations. In addition, these streaming databases use replication or upstream backup for recovery. These mechanisms require complex protocols. In contrast, CellIQ inherits the efficient parallel recovery mechanism from Spark.

Recently cellular network analytics systems have adopted the Hadoop based framework [5, 35]. However, they do not support efficient streaming graph computations. Since CellIQ's G-Stream abstraction unifies batch processing, graph processing and stream processing, it can support a range of processing models. For example, it can combine batch and stream processing by incorporating historical data in the analysis.

Temporal graph analytics systems Most large-scale graph processing systems have focused on static graphs. Some of these systems [19, 25, 32] can operate on multiple graphs independently. They do not expose an API or optimize for operations spanning multiple graphs. There are a couple of notable exceptions [9, 11, 21]. combBLAS [9] represent graphs and data as matrices and support generalized binary operators. Kineograph constructs incremental snapshots of the graph. Chronos optimizes the in-memory layout of temporal graphs and the scheduling of iterative computation on those graphs. Unlike CellIQ, they do not present a general API that supports incremental sliding window computation and graph reduce operations. More importantly, they are not optimized for cellular network analytics. Cellular network graphs present both spatial (i.e. graph structural) and temporal locality. CellIQ is designed specifically to leverage these characteristics to support efficient analysis.

Large-scale streaming Several recent systems [2, 14, 30, 31] support streaming computation with high-level APIs similar to D-Streams. However, they do not support streaming graph processing. One notable exception is the recent announcement of Naiad [30]'s GraphLINQ [28]. GraphLINQ intends to provide rich graph functionality within a general-purpose dataflow framework. Similar to CellIQ, it can operate on streams of vertices and edges. However, it is not optimized for cellular network analytics. As we have shown, optimizations that leverage the characteristics of cellular network can significantly improve performance. The techniques we presented can be incorporated in other frameworks. For instance, our spatial optimizations can benefit GraphLINQ.

9 Conclusion and Future Work

Current cellular networks lack a flexible analytics engine. Existing cellular data analytic systems are either elementary or lack support for real-time analytics. In this paper, we present CellIQ, an efficient cellular analytic system that can support rich analysis tasks. It represents cellular network data as a stream of property graphs. Leveraging domain specific knowledge, CellIQ incorporates a number of optimizations such as geo-partitioning of input data and co-partitioning of vertices and edges to reduce data movements, radius-based message broadcast for efficient spatial operations, incremental graph updates to avoid the cost of frequent joins in time window operations and differential graph updates for efficient sliding window operations. Our evaluations show that these techniques enable CellIQ to perform 2× to 5× faster compared to implementations that do not consider domain specific optimizations.

We see several arenas for future work. We are working on using CellIQ to perform root cause analysis on operational LTE networks. We would also like to explore the possibility of applying CellIQ's techniques on domains other than cellular networks. In this respect, we are working on streaming graph analysis techniques that are applicable to any arbitrary graphs.

[8]CellIQ assumes that the data is human generated in some of its optimizations which may not be always valid in the IoT domain.

Acknowledgments

Part of this work was performed while Anand was at Bell Labs, supported by NSF grant CNS 1218668. We thank Aurojit Panda, Shivaram Venkataraman and other AMPLab members who provided feedback on earlier drafts of this paper. We also thank our shepherd, Suman Banerjee, and our reviewers for their valuable feedback. This research is supported in part by NSF CISE Expeditions Award CCF-1139158, LBNL Award 7076018, and DARPA XData Award FA8750-12-2-0331, and gifts from Amazon Web Services, Google, SAP, The Thomas and Stacey Siebel Foundation, Adatao, Adobe, Apple, Inc., Blue Goji, Bosch, C3Energy, Cisco, Cray, Cloudera, EMC, Ericsson, Facebook, Guavus, Huawei, Informatica, Intel, Microsoft, NetApp, Pivotal, Samsung, Splunk, Virdata and VMware.

References

[1] Aggarwal, B., Bhagwan, R., Das, T., Eswaran, S., Padmanabhan, V. N., and Voelker, G. M. Netprints: diagnosing home network misconfigurations using shared knowledge. In *Proceedings of the 6th USENIX symposium on Networked systems design and implementation* (Berkeley, CA, USA, 2009), NSDI'09, USENIX Association, pp. 349–364.

[2] Akidau, T., Balikov, A., Bekiroğlu, K., Chernyak, S., Haberman, J., Lax, R., McVeety, S., Mills, D., Nordstrom, P., and Whittle, S. Millwheel: Fault-tolerant stream processing at internet scale. *Proc. VLDB Endow. 6*, 11 (Aug. 2013), 1033–1044.

[3] Alcatel Lucent. 9900 wireless network guardian. http://www.alcatel-lucent.com/products/9900-wireless-network-guardian, 2013.

[4] Alcatel Lucent. Alcatel-Lucent motive big network analytics for service creation. http://resources.alcatel-lucent.com/?cid=170795, 2014.

[5] Alcatel Lucent. Motive big network analytics. http://www.alcatel-lucent.com/solutions/motive-big-network-analytics, 2014.

[6] Arasu, A., Babcock, B., Babu, S., Datar, M., Ito, K., Nishizawa, I., Rosenstein, J., and Widom, J. Stream: The stanford stream data manager (demonstration description). In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA, 2003), SIGMOD '03, ACM, pp. 665–665.

[7] Bahl, P., Chandra, R., Greenberg, A., Kandula, S., Maltz, D. A., and Zhang, M. Towards highly reliable enterprise network services via inference of multi-level dependencies. In *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications* (New York, NY, USA, 2007), SIGCOMM '07, ACM, pp. 13–24.

[8] Balazinska, M., Balakrishnan, H., Madden, S. R., and Stonebraker, M. Fault-tolerance in the borealis distributed stream processing system. *ACM Trans. Database Syst. 33*, 1 (Mar. 2008), 3:1–3:44.

[9] Buluç, A., and Gilbert, J. R. The combinatorial BLAS: design, implementation, and applications. *IJHPCA 25*, 4 (2011), 496–509.

[10] Carney, D., Çetintemel, U., Cherniack, M., Convey, C., Lee, S., Seidman, G., Stonebraker, M., Tatbul, N., and Zdonik, S. Monitoring streams: A new class of data management applications. In *Proceedings of the 28th International Conference on Very Large Data Bases* (2002), VLDB '02, VLDB Endowment, pp. 215–226.

[11] Cheng, R., Hong, J., Kyrola, A., Miao, Y., Weng, X., Wu, M., Yang, F., Zhou, L., Zhao, F., and Chen, E. Kineograph: Taking the pulse of a fast-changing and connected world. In *Proceedings of the 7th ACM European Conference on Computer Systems* (New York, NY, USA, 2012), EuroSys '12, ACM, pp. 85–98.

[12] Cherniack, M., Balakrishnan, H., Balazinska, M., Carney, D., Cetintemel, U., Xing, Y., and Zdonik, S. Scalable Distributed Stream Processing. In *CIDR 2003 - First Biennial Conference on Innovative Data Systems Research* (Asilomar, CA, January 2003).

[13] Cohen, I., Goldszmidt, M., Kelly, T., Symons, J., and Chase, J. S. Correlating instrumentation data to system states: a building block for automated diagnosis and control. In *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6* (Berkeley, CA, USA, 2004), OSDI'04, USENIX Association, pp. 16–16.

[14] Condie, T., Conway, N., Alvaro, P., Hellerstein, J. M., Elmeleegy, K., and Sears, R. Mapreduce online. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation* (Berkeley, CA, USA, 2010), NSDI'10, USENIX Association, pp. 21–21.

[15] Cranor, C., Johnson, T., Spataschek, O., and Shkapenyuk, V. Gigascope: a stream database for network applications. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data* (New York, NY, USA, 2003), SIGMOD '03, ACM, pp. 647–651.

[16] Das, T., Zhong, Y., Stoica, I., and Shenker, S. Adaptive stream processing using dynamic batch sizing. In *Proceedings of the ACM Symposium on Cloud Computing* (New York, NY, USA, 2014), SOCC '14, ACM, pp. 16:1–16:13.

[17] Finkel, R., and Bentley, J. Quad trees a data structure for retrieval on composite keys. *Acta Informatica 4*, 1 (1974), 1–9.

[18] Gonzalez, J., Xin, R., Dave, A., Crankshaw, D., and Franklin, Stoica, I. Graphx: Graph processing in a distributed dataflow framework. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)* (Broomfield, CO, Oct. 2014), USENIX Association.

[19] Gonzalez, J. E., Low, Y., Gu, H., Bickson, D., and Guestrin, C. Powergraph: Distributed graph-parallel computation on natural graphs. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation* (Berkeley, CA, USA, 2012), OSDI'12, USENIX Association, pp. 17–30.

[20] Guttman, A. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA, 1984), SIGMOD '84, ACM, pp. 47–57.

[21] Han, W., Miao, Y., Li, K., Wu, M., Yang, F., Zhou, L., Prabhakaran, V., Chen, W., and Chen, E. Chronos: A graph engine for temporal graph analysis. In *Proceedings of the Ninth European Conference on Computer Systems* (New York, NY, USA, 2014), EuroSys '14, ACM, pp. 1:1–1:14.

[22] Kandula, S., Mahajan, R., Verkaik, P., Agarwal, S., Padhye, J., and Bahl, P. Detailed diagnosis in enterprise networks. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication* (New York, NY, USA, 2009), SIGCOMM '09, ACM, pp. 243–254.

[23] Karypis Lab, University of Minesota. Metis - serial graph partitioning and fill-reducing matrix ordering. http://glaros.dtc.umn.edu/gkhome/metis/metis/overview, 2014.

[24] Khanna, G., Yu Cheng, M., Varadharajan, P., Bagchi, S., Correia, M. P., and Verı́ssimo, P. J. Automated rule-based diagnosis through a distributed monitor system. *IEEE Trans. Dependable Secur. Comput. 4*, 4 (Oct. 2007), 266–279.

[25] Low, Y., Gonzalez, J., Kyrola, A., Bickson, D., Guestrin, C., and Hellerstein, J. M. Graphlab: A new framework for parallel machine learning. In *UAI* (2010), P. Gr¨1nwald and P. Spirtes, Eds., AUAI Press, pp. 340–349.

[26] Malewicz, G., Austern, M. H., Bik, A. J., Dehnert, J. C., Horn, I., Leiser, N., and Czajkowski, G. Pregel: A system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA, 2010), SIGMOD '10, ACM, pp. 135–146.

[27] McSherry, F., Murray, D. G., Isaacs, R., and Isard, M. Differential dataflow. In *CIDR 2013, Sixth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 6-9, 2013, Online Proceedings* (2013).

[28] Microsoft Naiad Team. GraphLINQ: A graph library for naiad. http://bigdataatsvc.wordpress.com/2014/05/08/graphlinq-a-graph-library-for-naiad/, 2014.

[29] MIT Technology Review. How wireless carriers are monetizing your movements. http://www.technologyreview.com/news/513016/how-wireless-carriers-are-monetizing-your-movements/, 2013.

[30] Murray, D. G., McSherry, F., Isaacs, R., Isard, M., Barham, P., and Abadi, M. Naiad: A timely dataflow system. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles* (New York, NY, USA, 2013), SOSP '13, ACM, pp. 439–455.

[31] Qian, Z., He, Y., Su, C., Wu, Z., Zhu, H., Zhang, T., Zhou, L., Yu, Y., and Zhang, Z. Timestream: Reliable stream computation in the cloud. In *Proceedings of the 8th ACM European Conference on Computer Systems* (New York, NY, USA, 2013), EuroSys '13, ACM, pp. 1–14.

[32] Roy, A., Mihailovic, I., and Zwaenepoel, W. X-stream: Edge-centric graph processing using streaming partitions. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles* (New York, NY, USA, 2013), SOSP '13, ACM, pp. 472–488.

[33] Sagan, H. *Space-filling curves*, vol. 18. Springer-Verlag New York, 1994.

[34] UC Berkeley. Berkeley Data Analytics Stack. https://amplab.cs.berkeley.edu/software/, 2014.

[35] Verizon. Verizon adds cloudera's cloud-based big data analytics solution to verizon cloud ecosystem. http://www.verizon.com/about/news/verizon-adds-clouderas-cloudbased-big-data-\analytics-solution-verizon-cloud-ecosystem/, 2013.

[36] Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M. J., Shenker, S., and Stoica, I. Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation* (Berkeley, CA, USA, 2012), NSDI'12, USENIX Association, pp. 2–2.

[37] Zaharia, M., Das, T., Li, H., Hunter, T., Shenker, S., and Stoica, I. Discretized streams: Fault-tolerant streaming computation at scale. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles* (New York, NY, USA, 2013), SOSP '13, ACM, pp. 423–438.