Query and Update Load Shedding With MobiQual

Dodda Sudeep¹, Bala Krishna²

¹Pursuing M.Tech(CS), Nalanda Institute of Engineering & Technology, Siddharth Nagar, Sattenapalli, Guntur., Affiliated to JNTUK, Kakinada, A.P., India.

²Asst. Professor, Department of Computer Science Engineering, Nalanda Institute of Engineering & Technology, Siddharth Nagar, Sattenapalli, Guntur., Affiliated to JNTUK, Kakinada, A.P., India.

Abstract - Freshness and accuracy are two key measures of quality of service (QoS) in location-based, mobile continual queries (CQs). However, it is often difficult to provide both fresh and accurate CQ results due to (a) limited resources in computing and communication and (b) fast-changing load conditions caused by continuous mobile node movement. Thus a key challenge for a mobile CQ system is: How do we achieve the highest possible quality of the query results, in both freshness and accuracy, with currently available resources under changing load conditions? In this paper, we formulate this problem as a load shedding one, and develop MobiQual—a QoS-aware framework for performing both update load shedding and query load shedding. The design of MobiQual highlights three important features. (1) Differentiated load shedding: Different amounts of query and update load shedding are applied to different groups of queries and mobile nodes, respectively. (2) Per-query QoS specifications: The overall freshness and accuracy of the query results are maximized with individualized QoS specifications. (3) Low-cost adaptation: MobiQual dynamically adapts, with a minimal overhead, to changing load conditions and available resources. We show that, through a careful combination of update and query load shedding, the MobiQual approach leads to much higher freshness and accuracy in the query results in all cases, compared to existing approaches.

Keywords—Load shedding, Query, MobiQual, QOS.

1.INTRODUCTION

Today, we are experiencing a world where we can stay connected while on-the-go because there are (1) myriads of affordable mobile devices and (2) ever increasing accessibility of wireless communications for these devices. Combined with the availability of low-cost positioning devices, like GPS, this has created a new class of applications in the area of mobile location-based services (LBSs). Examples include location-aware information delivery and resource management, such as transportation services, fleet management, mobile games, and battlefield coordination.

A key challenge for LBSs is a scalable location monitoring system capable of handling large number of mobile nodes and processing complex queries over their positions. Although several mobile continual query (CQ) systems have been proposed to handle long-running location monitoring tasks in a scalable manner [1], [2], the focus of these works is primarily on efficient indexing and query processing techniques, not on accuracy or freshness of the query results. Accuracy (inaccuracy) is defined based on the amount of mobile node position errors found in the query results at the time of query re-evaluation. This accuracy measure is strongly tied to the frequency of position updates received from the mobile nodes. Although one can also use a higher level concept to measure accuracy, such as the amount of containment errors found in the query results, including both false positives (inclusion errors) and false negatives (exclusion errors), we argue that using position update errors for accuracy measure will provide higher level of precision. This is primarily because by utilizing the amount of node position errors as the accuracy measure, one can easily bound the inaccuracy by a threshold-based position reporting scheme [3], [4].

Freshness (staleness), on the other hand, refers to the age of the query results since the last query re-evaluation. It is dependent on the frequency of query...
re-evaluations performed at the server. As mobile nodes continue to move, there are further deviations in mobile node positions after the last query re-evaluation. However, such deviations are not attributed to inaccuracy. Hence, freshness can be seen as a metric capturing the post-query-re-evaluation deviations in mobile node positions. It is important to note that higher freshness does not necessarily imply higher accuracy and vice versa. To obtain fresher query results, the CQ server must reevaluate the continual queries more frequently, requiring more computing resources. Similarly, to attain more accurate query results, the CQ server must receive and process position updates from the mobile nodes in a higher rate, demanding communication as well as computing resources. However, it is almost impossible for a mobile CQ system to achieve 100% fresh and accurate results due to continuously changing positions of mobile nodes. A key challenge therefore is: How do we achieve the highest possible quality of the query results in both freshness and accuracy, in the presence of changing available resources and changing workloads of location updates and location queries?

In this paper, we propose MobiQual—a resource-adaptive and QoS-aware load shedding framework for mobile CQ systems. MobiQual is capable of providing high-quality query results by dynamically determining the appropriate amount of update load shedding and query load shedding to be performed according to the application-level QoS specifications of the queries. An obvious advantage of combining query load shedding and update load shedding within the same framework is to empower MobiQual with differentiated load shedding capability, that is configuring query re-evaluation periods and update inaccuracy thresholds for achieving high overall QoS with respect to both freshness and accuracy.

2. LOAD SHEDDING IN MOBILE CQ SYSTEMS

In a mobile CQ system, the CQ server receives position updates from the mobile nodes through a set of base stations (see Figure 1) and periodically evaluates the installed continual queries (such as continual range or nearest neighbor queries) over the last known positions of the mobile nodes. Since the mobile node positions change continuously, motion modeling [3], [4] is often used to reduce the number of updates sent by the mobile nodes. The server can predict the locations of the mobile nodes through the use of motion models, albeit with increasing errors. Mobile nodes generally use a threshold to reduce the amount of updates to be sent to the server and to limit the inaccuracy of the query results at the server side below the threshold. Smaller thresholds result in smaller errors and higher accuracy, at the expense of a higher load on the CQ server. This is because a larger number of position updates must be processed by the server, for instance, to maintain an index [5]. When the position update rates are high, the amount of position updates is huge and the server may randomly drop some of the updates if resources are limited. This can cause unbounded inaccuracy in the query results. In MobiQual, we use accuracy-conscious update load shedding to regulate the load incurred on the CQ server due to position update processing by dynamically configuring the inaccuracy thresholds at the mobile nodes.

Another major load for the CQ server is to keep the query results up to date by periodically executing the CQs over the mobile node positions. More frequent query re-evaluations translate into increased freshness in the query results, also at the expense of a higher server load. Given limited server resources, when the rate of query re-evaluations is high, the amount of queries to be re-evaluated is vast and the server may randomly drop some of the re-evaluations, causing stale query results (low freshness). In MobiQual, we utilize freshness-conscious query load shedding to control the load incurred on the CQ server due to query re-evaluations by configuring the query re-evaluation periods. In general, the total load due to evaluating queries and processing position updates dominates the performance and scalability of the CQ server. Furthermore, the time-varying processing demands of a mobile CQ system entails that update and query load shedding should be dynamically balanced and adaptively performed in order to match the current workload with the server’s capacity.
3. THE MOBIQUAL APPROACH

The MobiQual system aims at performing dynamic load shedding to maximize the overall quality of the query results, based on per-query QoS specifications and subject to processing capacity constraints. The QoS specifications are defined based on two factors: accuracy and freshness. In MobiQual, the QoS specifications are used to decide on not only how to spread out the impact of load shedding among different queries, but also how to find a balance between query load shedding and update load shedding. The main idea is to apply differentiated load shedding to adjust the accuracy and freshness of queries. Namely, load shedding on position updates and query re-evaluations is done in such a way that the freshness and accuracy of queries are non-uniformly impacted.

From the perspective of query load shedding, we make two observations to show that nonuniform freshness in the query results can increase the overall QoS of the mobile CQ system: (1) Different queries have different costs in terms of the amount of load they incur. (2) Different queries have different tolerance to staleness in the query results. Thus it is more effective to shed load (by sacrificing certain amount of freshness) on a costly query than an inexpensive one. This is especially beneficial if the costly query happens to be less stringent on freshness, based on its QoS specification. Bearing these observations in mind, in MobiQual we employ QoS-aware query load shedding: We use query re-evaluation periods as control knobs to perform query load shedding, where the same amount of increase in query re-evaluation periods for different queries brings differing amounts of load reduction and QoS degradation with respect to freshness.

Similar to query load shedding, we make two observations regarding update load shedding to show that nonuniform result accuracy can increase the overall QoS. First, different geographical regions have different numbers of mobile nodes and queries. Second, different queries have different tolerance to position errors in the query results. This means that shedding more updates from a region with a higher density of mobile nodes and a lower density of queries can bring a higher reduction on the update load and yet have a smaller impact on the overall query result accuracy. This is especially true if the queries within the region have less stringent QoS specifications in terms of accuracy. Thus, in MobiQual we employ QoS-aware update load shedding: We use inaccuracy thresholds from motion modeling as control knobs to adjust the amount of update load shedding to be performed, where the same amount of increase in inaccuracy thresholds for different geographical regions brings differing amounts of load reduction and QoS degradation with respect to accuracy.

MobiQual dynamically maintains a throttle fraction, which defines the amount of load that should be retained. It performs both update load shedding and query load shedding to control the load of the system according to this throttle fraction, while maximizing the overall quality of the query results. MobiQual not only strikes a balance between freshness and accuracy by employing both query and update load-shedding, but also improves the overall quality of the results by utilizing per-query QoS specifications to capture each query’s different tolerance to staleness and inaccuracy.

3.1. Problem Formalization

The objective of the combined load shedding problem is to maximize the overall quality of

$$\Psi = \frac{1}{m} (\Psi_v + \Psi_u)$$

We now restate the processing constraint by combining the load due to query re-evaluation and update processing. Let \(z_v\) denote the fraction of the query load retained for a given set of re-evaluation periods \(\{P_j\}\). We have:

$$z_v = \frac{\sum_{i=1}^{k} f_c(C_j)/P_j}{\sum_{q \in Q} f_c((q))/\tau_q}.$$

Similarly, let \(z_u\) denote the fraction of the update load retained for a given set of inaccuracy thresholds \(\{\Delta_i\}\). We have:

$$z_u = \frac{\sum_{i=1}^{l} n_i \cdot f_r(\Delta_i)}{n \cdot f_r(\epsilon_i)}.$$

With these definitions, we can state the processing constraint as follows:
The parameter $\gamma$ in the above equation represents the cost of performing update processing with the setting of $i$, $\Delta_i = c_i$ compared to the cost of performing query re-evaluation with the setting of $j$, $P_j = \tau_j$. In other words, for the ideal case the query re-evaluations costs 1 unit, whereas the update processing costs $\gamma \in (0, \infty]$ units. Note that $\gamma$ is not a system specified parameter and is learned adaptively as follows. Let $U$ be the observed cost of update processing and $V$ be the observed cost of query re-evaluation during the last adaptation period. Then we have $\gamma = \frac{U}{zu} / \frac{V}{zv}$. This assumes that the workload does not significantly change within the time frame of the adaptation period. Recall that the load shedding parameters are configured after each adaptation period, thus yielding new values for $zu$ and $zv$ (by way of changing $P_j$’s and $\Delta_i$’s). Thus the combined load shedding problem is formalized as follows:

$$z_u + z_u \cdot \gamma \leq z \cdot (1 + \gamma)$$

3.2. The MQLS Algorithm

The basic principle of the MQLS algorithm is to start with the ideal case of $\forall j$, $P_j = \tau_j$ and $\forall i$, $\Delta_i = c_i$ and incrementally reduce the load to $z$ times that of the ideal case by repetitively increasing the re-evaluation period or the inaccuracy threshold that gives the smallest quality loss per unit cost reduction. The algorithm is greedy in nature, since it takes the minimum quality loss per cost step. Concretely, we partition the domain of re-evaluation periods and inaccuracy thresholds into $\beta$ segments, such that we increase the $P_j$’s and $\Delta_i$’s in increments of size $cv = (\tau - \tau_j)/\beta$ and $cu = (\epsilon - \epsilon_i)/\beta$, respectively. The MQLS algorithm maintains a min. heap that stores a $ qlpc$ value of a re-evaluation period (or an inaccuracy threshold) gives the quality loss per unit cost for increasing it by $cv$ units (or $cu$ units). The $qlpc$ value is denoted by $S^{qlpc}_j$ for query group $C_j$ and $S^{qlpc}_i$ for shedding region $A_i$. We have:

$$S^{qlpc}_j = \sum_{q \in Q} f_e(C_j) \cdot \frac{\frac{1}{P_j} V_j^r(P_j + c_u) - V_j^r(P_j)}{f_e(C_j) \cdot \frac{1}{P_j}}$$
$$S^{qlpc}_i = \gamma \cdot n_i \cdot f_r(\epsilon_i) \cdot \frac{U_i^r(\Delta_i + c_u) - U_i^r(\Delta_i)}{n_i \cdot (f_r(\Delta_i + c_u) - f_r(\Delta_i))}$$

The nominators of the second components in the above equations represent the changes in the quality due to the increment, whereas the denominators represent the changes in the cost. Note that the first components of the above equations are used to normalize the costs in the denominators, so that $S^{qlpc}_j$’s and $S^{qlpc}_i$’s can be compared.

When the MQLS algorithm starts, the current load expenditure of the system, which is the sum of the load due to update and query load shedding appropriately weighted by $\gamma$, is above our load budget imposed by the throttle fraction $z$. The algorithm iteratively pops the topmost element of the min. heap and depending on whether we have a re-evaluation period or inaccuracy threshold makes the increment using either $cv$ or $cu$. The $qlpc$ value of the popped element is updated based on above equations and is put back into the heap unless no further increments are possible. The algorithm runs until the load expenditure of the system is within the budget or all the re-evaluation periods and inaccuracy thresholds hit their maximum value. In the latter case the load cannot be shed to meet the processing constraint and random dropping of incoming updates as well as delay in query re-evaluations will unavoidably take place.

4. CONCLUSION

In this paper we presented MobiQual, a load shedding system aimed at providing high quality query results in mobile CQ systems. MobiQual has three unique properties. First, it uses per-query QoS specifications that characterize the tolerance of queries to staleness and inaccuracy in the query results, to maximize the overall QoS of the system. Second, it effectively combines query load shedding and update
load shedding within the same framework, through the use of differentiated load shedding concept. Finally, the load shedding mechanisms used by MobiQual are lightweight, enabling quick adaption to changes in the workload.

REFERENCES


AUTHORS PROFILE

Dodda Sudeep, Pursuing M.Tech(CS) from Nalanda Institute of Engineering & Technology, Siddharth Nagar, Sattenapalli, Guntur Affiliated to JNTUK, Kakinada, A.P., India. My research Interests are computer networks, Data Mining.

Bala Krishna, working as Asst. Professor, Department of Computer Science Engineering at Nalanda Institute of Engineering & Technology, Siddharth Nagar, Sattenapalli, Guntur Affiliated to JNTUK, Kakinada, A.P., India. My research Interests are Data Mining and Computer Networks.