# Parallel Evaluation of Composite Aggregate Queries

Lei Chen [#1], Christopher Olston [*2], Raghu Ramakrishnan [*3]

[#]*Computer Sciences Department, University of Wisconsin - Madison*
*1210 West Dayton Street, Madison, WI, 53705, USA*
[1]chenl@cs.wisc.edu

[*]*Yahoo! Research*
*2821 Mission College Blvd. Santa Clara, CA 95054 USA*
[2]olston@yahoo-inc.com
[3]ramakris@yahoo-inc.com

***Abstract*— Aggregate measures summarizing subsets of data are valuable in exploratory analysis and decision support, especially when dependent aggregations can be easily specified and computed. A novel class of queries, called composite subset measures, was previously introduced to allow correlated aggregate queries to be easily expressed.**

**This paper considers how to evaluate composite subset measure queries using a large distributed system. We describe a cross-node data redistribution strategy that takes into account the nested structure of a given query. The main idea is to group data into blocks in "cube space", such that aggregations can be generated locally within each block, leveraging previously proposed optimizations per-block. The partitioning scheme allows overlap among blocks so that sliding window aggregation can be handled. Furthermore, it also guarantees that the final answer is the union of local results with no duplication and there is no need for the expensive data combination step. We identify the most important partitioning parameters and propose an optimization algorithm. We also demonstrate effectiveness of the optimizer to minimize the query response time.**

## I. Introduction

Organizations rely heavily upon decision support tools that aggregate operational data. Aggregated data is also used, e.g., to construct prediction models, which are extensively used in online services such as item recommendation (Amazon.com) or targeted advertising (Yahoo.com). Such real-world tasks require the computation of multiple correlated aggregates.

We use a weblog analysis problem to illustrate the concept of queries with correlated aggregates. In this task, the goal is to understand users' behavior on the search results and the advertisements by analyzing massive web search session logs. The data have the following schema: (Keyword $k$, PageCount $p$, AdCount $a$, Time $t$). Each record indicates an instance that a user submitted a single keyword query $k$ at time $t$ and clicked $p$ search result links and $a$ associated advertisement links. We are interested in the correlation between the number of the result clicks and the ad clicks from different keyword groups during various time windows. Following measures have to be computed as a part of the analysis.

M1: For every minute and keyword, find the median of page count.
M2: For every hour and keyword, find the median of ad count.
M3: For every minute and keyword, find the ratio between M1 of that minute and M2 of the corresponding hour.
M4: For every keyword and a ten-minute sliding window, find the moving average of M3.

All the above measures are correlated with each other. Further, the results of all queries are required, not just the "final" measure of M4. In [4], we introduced a novel class of queries called *composite subset measures* to express a collection of correlated aggregates, including the dependencies among them. We also described an algorithm that can compute the aggregations altogether using single pass of sorting and scanning. In this paper, we will study how to evaluate such queries in a parallel manner; this is essential in order to scale to the data sizes we seek to handle.

A composite subset measure query can be evaluated by computing components one at a time, respecting the dependency ordering among those components. For the example analysis task, the query can be evaluated in the following manner:

Step 1. Repartition the data by keyword and minute, compute M1 in parallel.
Step 2. Repartition the data by keyword and hour, compute M2 in parallel.
Step 3. Perform parallel join over the results of M1 and M2, group the join results by keyword and hour, and compute the aggregated value of M3 for each group.
Step 4. Perform a sliding window aggregation over the result of M3 to generate the result of M4.

Such evaluation requires repartitioning and processing the raw data twice (Steps 1 and 2), which can be very expensive. Further, the results of M1 and M2 can be large, which makes the join operation costly. Instead, knowing that all the measure components should be evaluated together, we can use the following more efficient algorithm, which repartitions data only once and performs all the aggregations/joins locally within each partition block:

Step 1. Repartition data into blocks based on the keyword values.
Step 2. Compute M1–4 locally at each resulting block, using the previously proposed algorithm [4].
Step 3. Combine the local results as the final answer.

The degree of parallelism is limited by the number of keywords, and performance might level off if the number of available machines is high and the number of unique keywords is low. There is an alternative approach to evaluate this query:

Step 1. Repartition the raw data into sliding windows. Each window block contains data records for 25 consecutive hours. The last hour of a given block overlaps with the first hour of the next block.

Step 2. In each window block, compute M1–4 for the first 24 hours. Use the data in the last hour only to support the computation of the 24th hour.

Step 3. Merge the local results as the final answer.

In this overlapping data distribution scheme, each result block contains all the data required for some parts of the final answer. All the measures can be evaluated using one iteration of data redistributing and processing. Furthermore, the distribution granularity for this algorithm can be finer than that of the second algorithm, assuming there are more distinct days than the distinct keyword values. There will be more blocks, allowing more parallelism.

Finding a good data distribution scheme that guarantees the feasibility of local evaluation is a challenge. We also want to utilize work done for an operator for evaluating subsequent operators in the composite query workflow, as in [4]. Our approach uses the observation that composite measure queries group by regions in "cube space" induced by the dimension hierarchies. The data can be partitioned by region to allow localized evaluation.

### A. Contributions

Our first contribution in this paper is to identify what constitutes a feasible distribution, given the data schema and the query. We allow the result distribution blocks to overlap with each other, so that queries with sliding window aggregation component can be handled seamlessly. To our knowledge, this is the first work studying the overlapping data distribution scheme to handle correlated aggregations with sliding windows.

Our second contribution is to address the problem of finding the distribution scheme that minimizes the query response time. A cost model is presented to show how query response time is affected by the granularity of blocks as well as the overlap among blocks. An optimization algorithm is presented to search for the optimal values for the key parameters, and is evaluated using various metrics such as scalability, speed-up, and sensitivity to data skew.

### B. Paper Outline

The rest of the paper is organized as follows. We review the query model of composite subset measures in Section II. In Section III, we present the parallel evaluation algorithm based on feasible data distribution. We discuss the optimization problem in Section IV, and present the experimental results in Section VI. Sections VII and VIII cover the related work and the conclusion.

| **Name** | **Domains** |
|---|---|
| **K**eyword | word, group, ALL |
| **P**ageCount | value(0..200), level(low, medium, high), ALL |
| **A**dCount | value(0..20), level(low, medium, high), ALL |
| **T**ime | hour, day, month, year, ALL |

TABLE I
ATTRIBUTE DOMAINS

## II. CONCEPTS AND NOTATIONS

We now review the concept of composite subset measure queries, first introduced in [4]. Composite subset measure queries are applied to a table that is a bag (multiset) of records. Each attribute is associated with a set of hierarchical domains. For example, the values of the time attribute are drawn from the domains of minutes, hours, days, months, and years. Any value defined in a domain can be mapped to a unique value defined in any of the more general domains. For example, the value 02/07/2007 can be mapped into Feb. 2007 in month domain. The domain types can be either nominal or numeric. Table I shows domain descriptions for the sample analysis task. The special notation "ALL" refers to the most general domain which contains a unique value $ALL$, following [7].

Each data record can be mapped into a point in the space defined by the dimension attributes, called *cube space*. A *region* is a hyper-rectangle in cube space, and many natural regions arise because of the hierarchical value domains associated with dimensions. For example, ["baseball", 10, 10, 02/20/2007] is a region which represents all the records generated on Feb. 20, 2007 for keyword query "baseball," such that PageCount and AdCount are both 10. A data record is contained in region $r$ if its corresponding point in cube space is covered by the hyper-rectangle of $r$.

The *granularity* of a region is described in terms of the domains from which attribute values are drawn. For example, the granularity of region [SPORT, 10, 20, 02/20/2007] is <K:group, P:value, A:value, T:day> , assuming SPORT refers to a group of keywords. Those attributes defined in domain "ALL" can be omitted when the context is clear. So <K:keyword, P:ALL, A:ALL, T:ALL> is equivalent to <K:keyword>.

A region $r_1$ is the child region of another region $r_2$ if any data record contained in $r_1$ is also contained in $r_2$. Such "containment" implies the following properties: For any attribute, its domain in $r_1$ is more specific than that in $r_2$. The value of each attribute of $r_1$ can be mapped into the value of the corresponding attribute in $r_2$. $r_2$ is called the parent region of $r_1$.

Two regions are siblings if they share the same granularity in cube space. A specific set of siblings for a given region can be specified by adding range annotations to numeric attributes. For example, [badger, 20, 20, 2001:(0,3)] indicates the collection of siblings for region [badger, 20, 20, 2001]. They refer to regions with the same keyword, PageCount, and AdCount from year 2002 to 2004. Note that we cannot add an annotation to a nominal attribute because the meaning of "closeness" is not defined. The sibling set of a given region

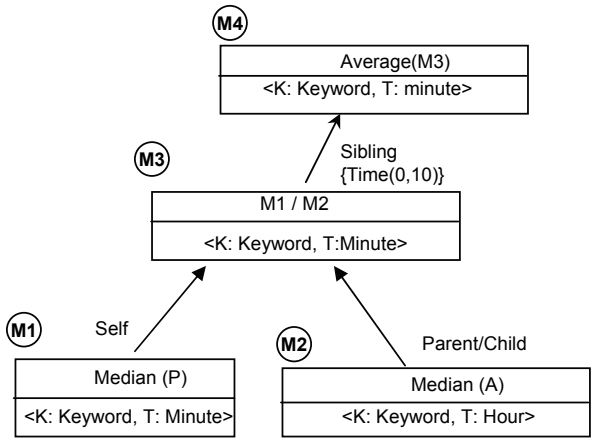



Fig. 1. Example Aggregation Workflow

| Relation | Meaning |
|---|---|
| **self** | The measure of a region is computed from the other measures of the same region |
| **child/parent** | The measure of a region is computed by aggregating the measures of its children |
| **parent/child** | The measure of a region is derived from the measure of its parent region |
| **sibling** | The measure is computed by aggregating the measures of sibling regions. The match condition for sibling is expressed as $\{X(l,h)\}$, where $X$ is the attribute name, (l,h) is the value range. |

TABLE II
MEASURE RELATIONSHIPS

can be defined using $\{X_i : (l_i, h_i)\}$, where $X_i$ is the attribute name and $l_i$ and $h_i$ are the lower and upper bounds.

A *region set* is the set of all regions with the same granularity, and can be uniquely identified using that common granularity. A region set is more general than the another region set if all attribute domains in the first region set are more general than the corresponding domains in the second region set.

### A. Query Model

A composite subset measure query consists of multiple correlated measure components. It can be expressed using a pictorial query language called aggregation workflows (see Figure 1). Each measure, illustrated as a node in the aggregation workflow, is defined over a region set and is an aggregation of data records contained in the region, or an aggregation of the result measures from related regions. The granularity and aggregation function for a given measure are shown in the node. The value dependencies among measures are shown as edges. Four relationships are used to establish the connections between measures. Table II summarizes the meaning of each relationship.

In an aggregation workflow, measures that do not depend on other measure components are called *basic measures*. Measures that depend on other measure components are called *composite measures*; the measures a composite measure depends upon are called its *sources*.

## III. PARALLEL EVALUATION ALGORITHM

### A. General Evaluation Strategy

In general, the parallel evaluation works as follows. The data records are stored in a distributed file in a machine cluster with shared-nothing architecture. Each file block has multiple replicas in the system to achieve better accessibility. The evaluation of a query involves two groups of machines: *mappers* and *reducers*. At the beginning of the evaluation, a given number of machines are assigned as the mappers and the reducers.

Each mapper fetches part of the datasets and generates key/value pairs from individual record. The $key$ field is computed by applying the transformation function on the data record, which will be described in Subsection III-B. The $value$ field is the exact copy of the original data record. Multiple key/value pairs might be generated from one single data record, which enables overlapped data redistribution.

The result pairs are shuffled and dispatched to reducers, with the constraint that those with the same key are dispatched to the same reducer. The reducers collect pairs and use external sorting to group pairs with the same key value. After the sorting, reducers process groups one at a time and produce aggregated results from each group. In [4], an algorithm was proposed to evaluate composite measure queries with centralized storage. This algorithm can be used as a subroutine to compute the local result from each group. The distribution scheme is designed in such a way that the final result is the simple union of local results without duplications.

Notice that we used the terms introduced in MapReduce [6], which is a large scale data processing framework for grouped aggregation. However, the algorithm can be implemented in any OLAP system which supports scatter-and-gather evaluation paradigm.

### B. Data Distribution Scheme

In order for the parallel algorithm to run correctly and efficiently, we set the following rules for a desirable data distribution scheme:

1. The union of the local results should be the final query answer.
2. There is no overlap among results generated from different blocks.
3. Overlaps among blocks are allowed, but should be minimized to ensure good performance.

In the rest of this section, we first consider the case when there is no sibling relation, which implies sliding window condition, in the query. This problem is relatively easy, because no overlap is required among the distributed blocks. Then we consider the more interesting case when sibling relations are used and overlapping data distribution might be needed, which is common when the query includes sliding windows.

*1) Non-Overlapping Distribution:* For multidimensional data set, it is straightforward to distribute the data records based on the grid of cube space. Data records which belong to the same region with a given granularity will be grouped

into the same distribution block. The region granularity serves as the *distribution key*, representing how the data records are grouped. We can efficiently identify the block a data record belongs to, by looking at the region that record resides in.

A distribution key is *feasible* if the value for any component measure in the given composite subset measure query can be evaluated based on the data records from a certain distribution block alone. For example, for the query in the example analysis task, <K:keyword> is a feasible distribution key since any measure can be computed from records with same keywords. On the other hand, <K:keword, T:minute> is not feasible, because the second measure (M2) is computed based on records from the entire hour.

To formally define the feasibility of a distribution key, we introduce the concept of the coverage set for a measure result. For a given measure result $mr =< r, v >$ , where $r$ is a region and $v$ is the associated measure value, the coverage set of $mr$, denoted as $coverage(mr)$, is the collection of data records that affect the value of $mr.v$. If a measure record is derived from other measures, then its coverage set is the union of the coverage sets for all source measure records. A distribution key $DK$ is feasible, if for any measure record in the result of the query, there exists a region $r_p$ with granularity $DK$ such that $coverage(mr) \subset r_p$. We have the following theorem about the feasible distribution key.

*Theorem 1:* If $X$ is a feasible distribution key and $X'$ is a generalization of $X$, then $X'$ is a feasible distribution key as well.

*Proof:*

Since $X$ is feasible, for any measure record $mr$, there exists a region $r_1$ with granularity $X$ such that $coverage(mr) \subset r_1$. $X'$ is the generalization of $X$, which means there exists a region $r_2$ with granularity $X'$ such that $r_1 \subset r_2$ for every $r_1$. This implies that $coverage(mr) \subset r_1 \subset r_2$. so $X'$ is a feasible distribution key as well. ■

The following theorem shows how to identify a feasible distribution key for composite subset measure queries without sibling relationships, based on the feasible distribution keys for its source measures. Furthermore, all the other feasible distribution keys can be derived from this particular key.

*Theorem 2:* If the composite subset measure does not include sibling relationships, then the least common ancestor of all measure granularities is a feasible distribution key. Furthermore, all the other feasible distribution keys are generalizations of this key, and the least common ancestor indicates a minimal key.

The proof of this theorem is in Appendix VIII.

Figure 2 shows how the data will be distributed based on key <K:keyword, T:hour>, which is feasible for the collection of M1, M2 and M3, but is not feasible if M4 is included.

*2) Overlapping Data Distribution:* For composite subset measure queries which contains sibling relations, the value of a measure record might depend on data records from its neighboring regions. Take M4 as an example, the result value is computed by combining the results from ten neighboring

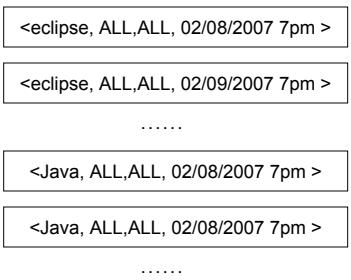



Fig. 2. Non-Overlapping Partition

regions with granularity <K:keyword, T:minute>. If we divide the data using distribution key <K:keyword, T:minute>, then regions with ten consecutive minutes might be assigned to different partition blocks and this moving window average measure can not be computed locally from any of these blocks.

One solution for such problem is to use a more general domain in the distribution key. For example, <K:keyword> is a feasible distribution key in which the time attribute is rolled up to the ALL domain. However, we can also solve this problem by allowing one distribution block to include multiple sibling regions with the same granularity. The union of such regions provides all the data records in the coverage set of the measure records in one given region. As a side effect, there will be data duplication among neighboring partition blocks, if we want all the aggregates to be computed locally.

The notation of the distribution key can be extended by attaching range annotations to individual attributes. The annotation indicates the number of preceding and succeeding regions that should be included in the distribution block. For example, <K:keyword T:minute(0,10)> is an extended distribution key. If we want to compute the measures defined in one region with granularity <K:keyword, T:minute> , the ten regions with the same keyword should be included in the same distribution block. Furthermore, the time value for these regions fall in the range of ten minute time window.

Formally, an overlapping distribution key can be defined using the following expression: $<X_1{:}D_1(l_1,h_1), \ldots, X_d{:}D_d(l_d,h_d)>$. $X_i$ is the attribute name. $D_i$ is the domain name. $l_i$ and $h_i$ are the lower bound and upper bound for the range annotation. If an attribute $X_i$ does not have the range annotation if $l_i = h_i = 0$.

In order to find the feasible distribution key for a given query, we can first identify the feasible distribution key for each individual measure component and then combine all the results keys together. Given the distribution keys for the sources measures and the range condition for sibling conditions, the feasible distribution key for the target measure can be derived.

We describe an algorithm with two basic operations to solve this problem. The first operation, called *opConvert*, takes the distribution key for the source measures and the sibling condition and produces a new distribution key, which is feasible for the target measure. The second operator, called *opCombine*, produces a distribution key based on the feasible distribution keys of all the source measures. The details of

| procedure **OpConvert**(distribution keys $k$, sibling condition $< x_1 : D_1(l_1, h_1), \ldots, x_d : D_d(l_d, h_d) >$) | |
|---|---|
| 1 | for each attribute $x_i$ |
| 2 | $D = domain(k.x_i)$ |
| 3 | $result.x_i.domain = D$ |
| 4 | $result.x_i.low = (k.x_i.low - map_D(h_i))$ |
| 5 | $result.x_i.high = (k.x_i.high - map_D(l_i))$ |

TABLE III
ADJUSTING DISTRIBUTION KEY USING SIBLINGS

| procedure **OpCombine**(distribution keys $k_1$, $k_2$, ...) | |
|---|---|
| 1 | if sibling relationship is used for the $i$th source |
| 1 | $k_i$ = opConvert($k_i$, sibling_condition) |
| 2 | for attribute $x$ |
| 3 | $D$ is the common generalization for domain($k_i.x$) |
| 4 | $k_i.x.low' = map_D(k_i.x.low), i = 1, 2, ...$ |
| 5 | $k_i.x.high' = map_D(k_i.x.high), i = 1, 2, ...$ |
| 6 | $low_0 = min\{k_i.x.low'\}$, $high_0 = max\{k_i.x.high'\}$ |
| 7 | $result.x.domain = D$ |
| 8 | if ($low_0 \neq 0$) or ($high_0 \neq 0$) |
| 9 | $result.x.low = low_0$ |
| | $result.x.high = high_0$ |

TABLE IV
CONSTRUCTING A FEASIBLE DISTRIBUTION KEY

these two algorithms are provided in Tables III and IV.

The *map* function is used in both operations, which converts the value range from one domain to the other domain. For example, the annotation T:day(-10,+60) can be converted into T:month(-1, +3). This is because a ten-day time window spans at most two month and a 60-day time window spans at most three months. The annotation of T:month(-1, 2) can be converted into T:day(-32,+63) following the similar reasoning.

The algorithm starts with basic measures. The feasible distribution key for each basic measure is the granularity of the measure. The distribution keys for composite measures are computed following the topological order of the aggregation workflow. For each composite measure, if the dependency is computed based on sibling relationships, *opConvert* is used to adjust the distribution key before the operation *opCombine* is applied. Otherwise, $opCombine$ will be applied directly.

Once the distribution key for each measure component is generated, the feasible distribution key of the whole query can be generated by combining the distribution keys for all the measure components using *opCombine*. The distribution key generated thus is "minimal," which means all the other feasible distribution keys for the query have the following property: if an attribute does not have range annotation in the minimal key, then the domains of that attribute in other feasible distribution keys are generalizations for the domain in the minimal key. If an attribute has a range annotation, then that attribute in all the other distribution keys either is defined in the ALL domain, or is defined in the same domain as that in the minimal key, and has a boarder range annotation.

When overlapping distribution is used, since there might be duplicated records among distribution blocks, the measure for a given region can be generated from the different blocks. Some of these results might not be correct, since not all the data records in its coverage set reside in the that block. The cost of removing the incorrect and duplicated results can be avoided if they can be filtered in the reducers. Such filtering is done by comparing the region of measure records to the value of the distribution key. In the key generation process, multiple key values might be generated from one single data record. However, there is a unique pair where the key is generated without being adjusted with a delta value. As a result, we only output a measure record in the reducer when its associated region resides in the region specified by the current group.

### C. Reducing Redundant Data

When we choose to use an overlapping distribution key, there are data duplicates between two neighbor blocks. Usually, these two blocks will be assigned to different reducers and the duplicated portion has to be transferred and processed twice. In the extreme example of query M4, if we use the distribution key <K:keywourd, T:minute(0,10)>, one data record will essentially be duplicated in ten blocks. The total workload will be increased approximately by a factor of ten. Such data duplication can be reduced by forcing neighbor blocks into one. As the size of each block increases, the portion of duplicated data is reduced. The number of distribution blocks to be merged together is called *clustering factor*.

Figure 3 shows the effect of such overlapping distribution. The original distribution key is $< T : day(0, 2) >$. The gray regions represent those from which the measure results need to be calculated. The white blocks are regions which are assigned to the distribution block to provide input data for the measures in gray regions. No result will be output from those white regions. Figure (a) shows the case when clustering factor equals 1 and (b) shows the case when the clustering factor equals 2. The original dataset consists of six regions. In case (a), the number of total regions is 12 and for case (b) the number of such regions is 8. It is clear that scheme in case (b) results in lower workload.

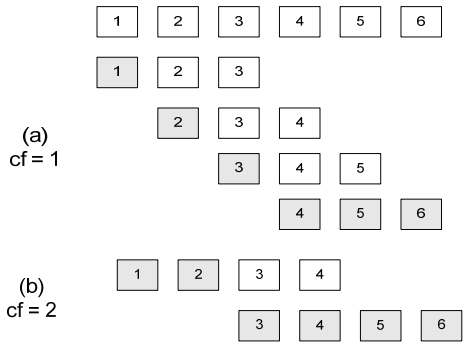



Fig. 3. Redistribution with Clustering Factor

Certain modification has to be made in the key generation logic to support such composite blocks. The essential idea is to use one unique value to represent a range of values for

a given attribute. Suppose time is the annotated attribute in the distribution key. For a given record, the time value is 100 and the clustering factor equals to 5, then the time value in the key should be converted to $100 \text{ div } 5 = 20$. After the division, the same result will be derived from values between 100 to 104, hence those regions with neighboring time values will be assigned with the same key value and hence be dispatched to the same distribution block. Since only numeric attribute is allowed to have range annotation, such value conversion is always feasible.

By using larger value for the clustering factor, the amount of data to be duplicated will decrease. This will reduce the volume of data records to be transferred and processed. However, such distribution scheme has one side effect: there are fewer resulting blocks. This limits the degree of parallelism for the computation. We provide an detailed analytical model in Subsection IV-B.

### D. Other Implementation Considerations

We discuss two additional considerations in this subsection. First, in the parallel evaluation strategy, the size of data to be transferred is at least as large as the original data set. In certain situations the data size can be reduced by performing early aggregation for the basic measures and transferring the aggregation results instead. This strategy will be effective when (1) The number of basic measures is not very high (2)There is substantial size reduction from the raw data to the results of basic measures (3)All the basic measures are either algebraic or distributive so that partial results can be generated in the early stage of the computation. In such situations it makes sense to perform early aggregation. We measure the performance impact on using early aggregation in conjunction with our evaluation strategy in Section VI.

Secondly, the way MapReduce forms groups at each reducer is by sorting the data it receives based on the distribution key. Once the MapReduce system performs this sort, it hands the content of each group to a user-provided reducing function. Our algorithm's reducing function immediately re-sorts the content of each group, according to the sort key that has been chosen by the local sort/scan algorithm. Obviously, the two sort passes could be combined into one (using a composite sort key). A substantial amount of work can be saved, especially if the data volume is such that out-of-core sorting is necessary. Our current implementation uses an unmodified Map/Reduce implementation that does not support this functionality. The quantification of the performance impact of such optimization is presented in Section VI.

## IV. Distribution Scheme Optimization

In this section, we study the problem of finding the optimal distribution key and clustering factor to minimized the evaluation response time. We focus on the response time since it is the main factor that affects the efficiency of an interactive analysis.

During the evaluation of a composite subset measure query, the data records are distributed among multiple reducers. The response time for a specific execution plan is due to the following components: (1)fetching data records in each mapper, (2)transferring key/record pairs to the reducers, and (3)processing the data in the reducer using the local sort/scan algorithm. The first cost is not affected by the distribution scheme. The second and third factors are proportional to the heaviest workload assigned to an individual reducer. The total response time can be reduced by minimizing the heaviest workload.

There are two possible ways to reduce the heaviest workload. First, we can reduce the data which are duplicated among neighbor distribution blocks. We prefer to use the non-overlapping distribution key to avoid processing duplicates. Second, it is desirable to have blocks evenly assigned among different reducers so that the expected value of the heaviest workload is minimized. More detailed analysis will be provided in the following subsections.

### A. Non-Overlapping Distribution Keys

We first analyze the case when a non-overlapping distribution key is used. In this case, each distribution block contains a single region. The distribution key corresponds to a region granularity $G$. $n_G$ is the total number of regions with granularity $G$ in cube space. $m$ is the number of available reducers. We use $X_i$ to represent the number of data records within each region. $D_G$ is the distribution function of $X_i$, $X_i \sim D_G$. $B_i \in 1, 2, ..., m$ indicates the reducer that region $i$ is assigned to. Then, the number of data records assigned to a given reducer $j$ is $\sum_{i=1...n_G} X_i \times I(B_i = j))$, where $I(B_i = j)$ is the indicator function. If we assume regions are randomly assigned to reducers following a uniform distribution, the size of the heaviest workload assigned to an individual reducer is:

$$max_{j=1...m}(\sum_{i=1...n_G} X_i \times I(B_i = j)) \quad (1)$$

where $P(B_i = v) = 1/m$, for $v = 1, 2, ..., m$.

In general, it is hard to find the minimal value of Formula (1), because the form of $D_G$ is unknown. However, if we assume data records are distributed evenly in cube space, and hence each region contains exactly $N/n_G$ data records, where $N$ is the total number of data records, $X_1 = X_2 = ... = N/n_G$. The expected value for the size of heaviest workload is then $E(max_{j=1...m}(\sum_{i=1...n_G} I(B_i = j))) \times N/n_G$. When the values of $N$ and $n_G$ are fixed, the term $E(max_{j=1...m}(\sum_{i=1...n_G} I(B_i = j)))$ is exactly the first moment of the largest order statistic for a multinomial distribution $M(m, n_G)$. According to [9], [10], this can be approximated by Formula 2(the constant parameter $\alpha = 0.5772$) when the value of $n_G \to \infty$.

$$\frac{N}{m} + \sqrt{\frac{N^2}{n_G \times m} \times [2 \ln m - \frac{(\ln(\ln m)) + ln(4\pi) - 2\alpha}{2\sqrt{2 \ln m}}]} \quad (2)$$

This approximate formula will monotonically decrease as the value of $n_G$ increases. As the result, when two distribution keys are both feasible for the given query, a good strategy is to pick the one which is more specific in the granularity hierarchy and produces more regions with smaller sizes. As the number of blocks increase, the variation among the workloads assigned to different reducers decreases. Hence, the expected value of the heaviest workload assigned to an individual reducer will decrease as well. Using small regions also brings another benefit in the evaluation: sorting the data records within each block, which is required by the local algorithm, can be carried out inside memory without extra I/O.

Based on the above analysis and Theorem 2, we can draw the following conclusion: For non-overlapping distribution cases, the optimal distribution key is *the granularity of the least common ancestor for the region sets of all components in the composite subset measure query,* if distribution blocks are randomly assigned to the reducers and the data records are uniformly distributed in cube space.

### *B. Overlapping Distribution Keys*

Next, we consider overlapping distribution keys where one attribute in the key is associated with a range annotation. We first study the case when the basic granularity of the distribution key is fixed and the value of the clustering factor needs to be determined. The clustering factor indicates the number of neighbor distribution blocks to be merged together. Larger values for clustering factor will reduce the extra data duplication among neighbor blocks. However, if the value of clustering factor is too large, there are too few distribution blocks. As a result, some reducers might be under-utilized, which causes overall performance to deteriorate.

Suppose $G$ is the granularity indicated by the distribution key, $d$ is the difference between the lower bound and the upper bound of the range annotation. In order to guarantee the feasibility of local evaluation, there should be at least $d+1$ neighbor regions with granularity $G$ in each distribution block. Suppose the value of the clustering factor is $cf$. Then each distribution block will contain $d+cf$ regions and the number of data records within each block follows the distribution $D_G^{d+cf}$. The total number of distribution blocks is $\lceil (n_G - d)/cf \rceil$, which can be approximated by $\lceil n_G/cf \rceil$, when the value of $n_G$ is significantly larger than $d$. We use $X_i$ to represent the number of data records within block $i$ and $B_i$ to represent the reducer which block $i$ is assigned to.

If we assume that blocks are to be randomly assigned to reducers, following the uniform distribution, and data records are evenly distributed in cube space, Formula (3) is the expected value of the size of the heaviest workload:

$$E(max_{j=1\ldots m}(\sum_{i=1\ldots\lceil n_G/cf\rceil} X_i \times I(B_i = j))) \qquad (3)$$

where $X_i = N(d+cf)/n_G, i = 1, 2, \ldots, \lceil n_G/cf \rceil$ and $P(B_i = v) = 1/m$, for $v = 1, \ldots, m, i = 1, \ldots, \lceil n_G/cf \rceil$.

Formula (4) gives the approximate value for the heaviest workload assigned to a reducer. Compared with Formula (2), this formula is obtained by replacing $N$ with $N(d+cf)/cf$ and $n_G$ with $n_G/cf$. The first replacement indicates that we need to transfer more data due to the data replication caused by the overlapping distribution and the second replacement indicates that the number of blocks in the distribution is reduced. The optimal value of $cf$ can be found by forcing the derivative of (4) to be zero, which results in a cubic equation. The optimal value for $cf$ is either the floor or the ceiling of the root value of the resulting cubic equation. We omit a detailed evaluation of $cf$ due to lack of space.

$$N \times (d+cf) \times [(\frac{1}{m \times cf}) + \sqrt{\frac{1}{n_G \times m \times cf}} \times \sqrt{2\ln m - \frac{(\ln(\ln m)) + ln(4\pi) - 2\alpha}{2\sqrt{2\ln m}}}] \qquad (4)$$

In the above discussions, we assume the region granularity $G$ is already identified. The optimal region granularity can be found as follows: In Subsection III-B.2, we show that a "minimal" overlapping distribution key can be generated from the composite subset measure query with multiple alternative range annotations. In order to find the distribution key with single annotated attribute, the optimization algorithm will try to keep one attribute annotated at a time and roll up the domains of all the others into ALL. For each candidate key, the clustering factor that minimizes Formula (4) will be identified and the value of the expected workload computed. The combination of distribution key and clustering factor that minimizes the heaviest workload will be used in the execution plan.

## V. Handling Data Skew

The analyses for both non-overlapping and overlapping distribution keys are based on the assumption of uniform data distribution. When this assumption does not hold, we might not be able to get the optimal parameter values. This subsection presents run-time solutions to detect and handle data skew.

The first problem is to detect the existence of data skew. Notice that in the evaluation process, the mappers have to acquire the raw data records. Once the data records are acquired, we can first perform a simulated data dispatch. Each mapper can sample some records for the data it acquires, and compute the destination reducers. The workload assigned to each reducer is then computed. After that, the mappers can send the workload information to a single machine, which combines all this information and estimates the final workload for each reducer. If the workload of a reducer is significantly higher than the other reducers, this indicates the presence of data skew.

For queries without sibling conditions, the distribution key is minimal. All the other possible feasible distribution keys will use coarser granularities than the identified distribution

key. If the data distribution based on the minimal key is still not good enough, it is unlikely that other distribution schemes will perform better. In this case, data skew cannot be resolved directly using region-based data distribution.

For queries that require overlapping distribution keys in the evaluation process, the use of a larger clustering factor makes the algorithm more sensitive to data skew. There are two approaches to finding the appropriate value of the clustering factor. The first is heuristical. We impose a lower limit on the number of distribution blocks assigned to each reducer, and avoid distribution schemes in which the estimated number of distribution blocks assigned to each individual reducer is smaller than a predetermined value $X$.

In the second solution, instead of using the "optimal" distribution key found by the optimizer(Subsection IV-B), we modify the logic of the optimizer to output a list of candidate distribution keys. The candidate distribution keys are diversified in the sense that they have different annotated attributes or significantly different values of the clustering factor. Simulation data dispatches are performed based on each of the candidate keys, the key which results in lowest maximal workload will be selected in the evaluation.

It is worth noticing that the goodness of the distribution key is not bound with specific composite queries since it only affects how the raw data are distributed. As long as the value distribution of original data set does not change, a distribution key, which was previously identified as a good one, will still be good candidate, as long as it is feasible for the given query. That means we can store the good distribution keys used before, check whether it is feasible for a given query. and reuse the key in the evaluation of the current query.

## VI. Evaluation Result

Our experiments utilize a cluster of 100 machines, running open-source Map/Reduce implementation called Hadoop [8]. Each machine is equipped with a 2GHz Intel Xeon CPU, 4GB memory and two hard disks with 7200rpm speed and 200GB capacity. Up to two map or reduce tasks can be assigned to a given machine. The amount of memory available to an individual task is 800MB. The system maintains three replicas of each file, for fault tolerance.

It should be pointed out that although the algorithm is implemented and evaluated on the Hadoop platform, it can also be implemented on other parallel OLAP systems. Due to the simple evaluation logic, we expect to see similar performance, with the main difference coming from different implementations of sorting and data transferral.

We use synthetic data sets that conform to the following schema: There are four integer attributes drawn from $[0, 255 \cdot (4^4 - 1)]$ with domain hierarchy with four levels, and two temporal attributes. The domain hierarchy for each of the temporal attributes is: $second$, $minute$, $hour$ and $day$. The domain of each temporal attribute spans a twenty-day period.

Two types of data distributions are used: uniform distribution and skewed distribution with temporal attributes. For the skewed case, the values of the temporal attributes are picked from the first five days of the twenty-day range.

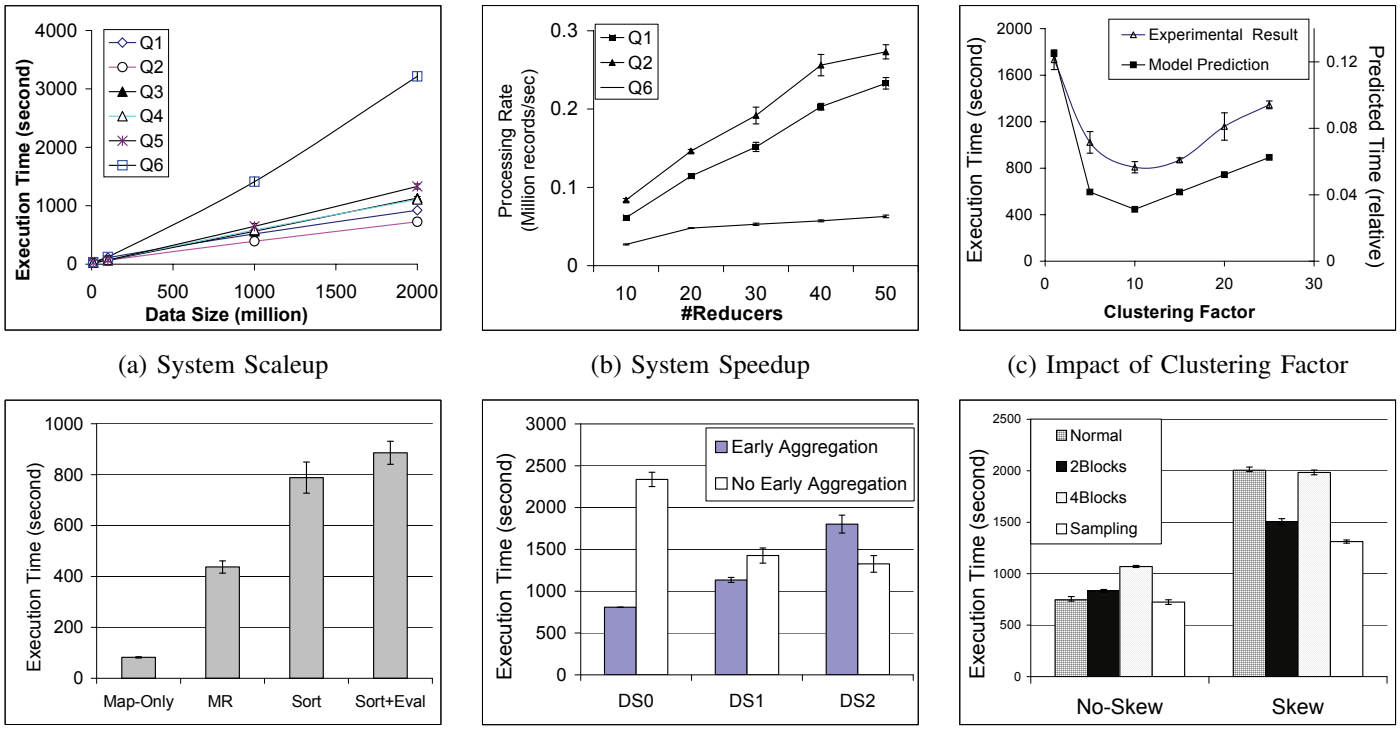
Six queries are used in the evaluation, labeled Q1-Q6. Q1 includes three independent measures defined over different region sets with fine granularities. Q2 includes two measures where the measures for parent regions are generated from those of the children regions. Q3 includes five measures; the measures for parent region sets are the aggregations of two different measures, both of which are computed by aggregating measures of their children region set. Q4 includes a measure which is computed by combining the measure for the same region and children regions. Q5 utilizes the sibling relations in the query. The composite measure for each hour is the summary for the measures of the previous hours. Q6 contains the mixture of all four relations with a sliding time window aggregation as the top measure of the query.

Figure 4(a) shows the query response time for date sets with different sizes. Each execution uses 50 mappers and 50 reducers. For all the queries, we observe that the execution time increases close to linearly with respect to the size of the input dataset. The evaluation of Q6 is consistently slower than other queries. It is due to the reason that there is a sibling relationship with range in Q6. We have to use overlapping distribution key which increase the data to be processed. The sorting within each distribution block is also comparatively slower because of the large block size.

Figure 4 (b) shows the average data processing rate (data set size divided by the response time) when we use the dataset with one billion records and linearly increase the number of mappers and reducers. The performance curve for Q1 and Q2 indicates that the system performance increases linearly as more machines are added. The performance for Q3–5 is similar to those for Q1–2 and is not shown in the figure. The curve for Q6 is not as good as the other two queries, this is because Q6 contains a large siding window with very coarse granularity and results in smaller clustering factor and more data overlapping among blocks.

Figure 4 (c) shows the execution time as we vary the value of the clustering factor. The naive scheme that performs no clustering (clustering factor $= 1$) is about twice as slow as the optimal scheme (clustering factor $= 10$). However, an excessively large clustering factor (e.g. clustering factor $= 25$) also performs poorly, because the opportunity for parallelism is reduced. Hence it is crucial to choose an appropriate clustering factor. In the figure, we overlay our analytical prediction from Section IV-B. Note the close similarity between the actual performance curve and the predicted one. Hence, our model can be used to predict a good clustering factor. As the figure shows, the optimal value for the clustering factor is 10.

Figure 4 (d) shows the breakdown of the cost for the different parts of the evaluation. *Map-Only* represent the case when we only fetch data via the mappers. *MR* represents the case when the data are sent to reducers and sorted by the mapping key. *Sort* represents the case when the data are sorted within each group and *Sort+Eval* represents the case when the sorted data are scanned and result generated. The




(a) System Scaleup (b) System Speedup (c) Impact of Clustering Factor

(d) Evaluation Cost Breakdown (e) Effect of Early Aggregation (f) Handling Data Skew

Fig. 4. Experiment Results

cost of the first case is very low, which shows the feasibility of our run-time solution to handle data skew. There is a significant gap between the second case and the third case, such cost is associated with the data sorting within each group. As discussed in Section III-D, this cost can in principle be eliminated if the evaluation framework supports features to combine the distribution key and the key for the local algorithm together. Finally, the difference between the last two cases is very small, which implies the low overhead for scan evaluation.

To illustrate the effectiveness of doing early aggregation in the mapper, we measure the performance of three composite subset measure queries, labeled DS0-DS2. Each query consists of a basic measure and several composite measures. We constructed the queries such that they differ in the granularity of grouping. Query DS0 uses a very coarse granularity of grouping, whereas DS2 uses a fine granularity. DS1 uses an intermediate granularity.

Figure 4 (e) shows the performance comparison with early aggregation with data sets containing two billion records. The performance impact of early aggregation depends on the granularity of the basic measure. When the basic measure is defined on a very coarse level (DS0), there is significant size reduction, and early aggregation is clearly advantageous. The advantage decreases when the basic measure is defined at a finer granularity, as in DS1. For the last case (DS2), the performance of the algorithm with early aggregation is actually worse than without early aggregation. In this case, the overhead of performing early aggregation (which requires a sort or hash on the mappers) outweighs the benefit.

Figure 4 (f) shows results for handling data skew. We use a data set with one billion records. The "With-Skew" data set follows a skewed distribution over the temporal attribute and the "No-Skew" data set follows the uniform data distribution. *Normal* is the evaluation plan generated by the unmodified optimizer. The performance of this plan is compared against those of plans which enforce minimal number of estimated distribution blocks for each reducers (*2Blocks* and *4Blocks*), as well as plans performing run-time data sampling and simulation dispatching (*Sampling*). By imposing the lower bound, the optimizer might be able to find a better plan when data distribution is skewed. However, it can also be too conservative and pick plans with too much data overlap (4Blocks). In this case, the performance will be inferior when there is no data skew. On the other hand, the sampling approach is capable of finding the optimal evaluation plan with or without skew. The cost of the sampling is about 100 seconds for each data set, and is acceptable when compared to the overall evaluation cost and the performance boost achieved by using the data redistribution parameter.

## VII. Related Work

Most of the prior work either dealt with localized evaluation of correlated aggregation or parallel evaluation of uncorrelated aggregations.

[1] presented general strategies to compute multiple GroupBy aggregations. It showed that multiple aggregates can benefit from the same sorting order for the input dataset. The

solution did not cover correlated aggregates with complex match conditions. [5] provided an optimization framework to evaluate GroupBy queries with different attribute sets. The solution only considers simple containment relations (children/parent) among group set attributes.

[3] proposed a groupwise evaluation strategy that can identify the subquery from a SQL query which can be evaluated by partitioning the data based on grouping attribute. The groupwise strategy only supported equi-join and thus required the partition to be non-overlapping. Our algorithm is able to recognize the more complicated scenarios with sliding window conditions, by using overlapping data distribution.

[11] proposed adaptive algorithms that switch between aggregation algorithms based on the grouping selectivity. One of the algorithm, Repartition, is similar to the one proposed in this paper, where data records in the same region are mapped to the same machine; however, they do not provide details about how the partition scheme is identified.

[2] discussed a system that evaluates aggregation queries in a distributed data warehouse. Data reduction is performed locally and aggregated results are sent to the centralized coordinator. Such strategy is applied to the scenario when the communication cost is high. We are dealing with a different application, where relocating the raw data can be done with a reasonable cost. By redistributing the data, we can evaluate very complicated correlated aggregations.

## VIII. Conclusions

The following contributions have been made in this paper:

1. We identified a general strategy to evaluate correlated aggregation queries in parallel by redistributing data into blocks and computing results per-block. Overlapping redistribution scheme is supported in order to handle queries that contain sliding window component.
2. We implemented the algorithm using the MapReduce framework, which runs over clusters with hundreds of machines.
3. We developed an analytical model to measure the impact of key parameters over the query response time. Techniques were identified to find the optimal values for these parameters. The evaluation results clearly demonstrate the efficiency of our approach for evaluating queries with high parallelism.

## References


[1] S. Agarwal, R. Agrawal, P. Deshpande, A. Gupta, J. F. Naughton, R. Ramakrishnan, and S. Sarawagi. On the computation of multidimensional aggregates. *VLDB'96* , pages 506–521. 1996.

[2] M. O. Akinde, M. H. Böhlen, T. Johnson, L. V. S. Lakshmanan, and D. Srivastava. Efficient olap query processing in distributed data warehouses. *Information System*, 28(1-2):111–135, 2003.

[3] D. Chatziantoniou and K. Ross. Partitioned optimization of complex queries. *Information System*, 32(2):248-282, 2007.

[4] L. Chen, R. Ramakrishnan, P. Barford, B.-C. Chen, and V. Yegneswaran. Composite subset measures. In *VLDB' 06*, pages 403–414. VLDB Endowment, 2006.

[5] Z. Chen and V. Narasayya. Efficient computation of multiple group by queries. In *SIGMOD' 05*, pages 263–274. ACM Press, 2005.

[6] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI' 04*, pages 137–150, 2004.

[7] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and subtotal. *ICDE' 96*, pages 152–159., 1996.

[8] Hadoop project. http://lucene.apache.org/hadoop/. 2007.

[9] D. B. Owen and G. P. Steck. Moments of order statistics from the equicorrelated multivariate normal distribution. *The Annals of Mathematical Statistics*, 1962.

[10] M. Petzold. A note on the first moment of extreme order statistics from the normal distribution http://www.handels.gu.se/epc/archive/00001190/, 2000.

[11] A. Shatdal and J. F. Naughton. Adaptive parallel aggregation algorithms. In *SIGMOD '95*, pages 104–114, New York, NY, USA, 1995. ACM Press.


## Appendix: Proof of Theorem 2

*Proof:* For a basic measure, the region granularity associated with the measure represents a feasible distribution key, because the measure is computed by aggregating records within the region for given granularity.

When a value of a measure depends on multiple source measures, the coverage set of the original measure region is the union of those for the source measure regions. Without the sibling condition, the smallest region which contains all the coverage sets is the one with the least common ancestors granularity for both the original region set and the source region sets.

Finally, we have multiple component measures $m_1$, $m_2$, ..., $m_n$ and the feasible partition keys for individual component is $k_1$, $k_2$, ... $k_n$. As the Theorem 2 shows, the common ancestor of all the distribution keys is a feasible distribution key for every measure, and hence is the feasible distribution key for the entire aggregation workflow.

Suppose there exists another distribution key $X$ which is not a generalization of the least common ancestor $X_0$, there must exists an attribute $a$ which is defined on more general domain in $X_0$ than in $X$. Since $X_0$ is the least common ancestor, there exists a measure component $m'$ such that the domain of $a$ for the feasible distribution key of $a$ is defined in the domain as specified in $X_0$. So $X$ is not a feasible distribution key for $m'$. So such $X$ does not exists and $X_0$ is the "minimal" distribution key. ■