Correlation-Aware Object Placement for Multi-Object Operations*

Ming Zhong† Kai Shen Joel Seiferas
Department of Computer Science, University of Rochester
Email: {zhong, kshen, joel}@cs.rochester.edu

Abstract

A multi-object operation incurs communication or synchronization overhead when the requested objects are distributed over different nodes. The object pair correlations (the probability for a pair of objects to be requested together in an operation) are often highly skewed and yet stable over time for real-world distributed applications. Thus, placing strongly correlated objects on the same node (subject to node space constraint) tends to reduce communication overhead for multi-object operations. This paper studies the optimization of correlation-aware data placement. First, we formalize a restricted form of the problem as a variant of the classic Quadratic Assignment problem and we show that it is NP-hard. Based on a linear programming relaxation, we then propose a polynomial-time algorithm that generates a randomized object placement whose expected communication overhead is optimal. We further show that the computation cost can be reduced by limiting the optimization scope to a relatively small number of most important objects. We quantitatively evaluate our approach on keyword index placement for full-text search engines using real traces of 3.7 million web pages and 6.8 million search queries. Compared to the correlation-oblivious random object placement, our approach achieves 37–86% communication overhead reduction on a range of optimization scopes and system sizes. The communication reduction is 30–78% compared to a correlation-aware greedy approach.

1 Introduction

In distributed data-intensive applications, sometimes a user-level operation needs to access multiple data objects. For example, in a full-text keyword search engine using pre-constructed keyword indices, a multi-word search request requires access to multiple keyword indices. Similarly, an aggregation query accesses multiple data objects in a distributed database. When the requested objects are placed at different nodes, a multi-object operation typically incurs communication overhead for moving data and synchronizing actions. As demonstrated by the example in Figure 1, the choice of the object placement strategy can substantially affect such communication overhead. Specifically, Figure 1(a) suggests that placing correlated objects on the same node makes more requests locally computable. In comparison, another object placement scheme in Figure 1(b) incurs substantially higher communication overhead due to its oblivion to object correlations.

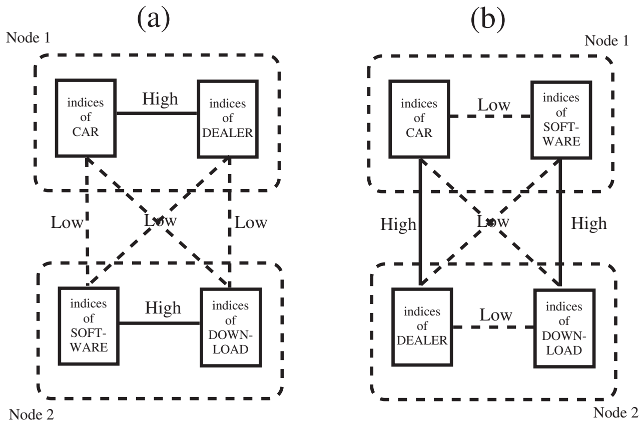

Figure 1. An illustration on the importance of object placement strategies in distributed full-text keyword search. Here "CAR, DEALER" and "SOFTWARE, DOWNLOAD" are highly correlated object pairs (requested together with relatively high frequency).

We define the *correlation* between a pair of objects as the probability for them to be requested together in any given operation. As explained above, placing more correlated objects on the same node tends to reduce required communication overhead. However, existing object place-

*This work was supported in part by the U.S. National Science Foundation grants CCR-0306473, ITR/IIS-0312925, CAREER Award CCF-0448413, CNS-0615045, CCF-0621472, and by an IBM Faculty Award.

†Zhong is currently affiliated with Google.

ment approaches in distributed data-intensive applications are often oblivious to object correlations in multi-object operations. At the best, some correlation-aware heuristic approaches may be employed for object placement with no understanding on their optimality. In principle, a desirable object placement strategy must consider complex inter-object correlations while it is also subject to constraints such as the available space of each node. In this paper, we address the problem of finding the optimal object placement strategy that minimizes the total communication cost for multi-object operations with known object correlations, object sizes, and per-node space constraints.

The remainder of this paper is organized as follows. Section 1.1 discusses motivating applications to our studied problem. In Section 2, we formalize our optimization problem and show its NP-hardness. Based on a linear programming relaxation, we propose a polynomial-time approximation algorithm that generates a randomized object placement scheme with an optimal *expected* communication overhead. We also describe related work to our problem. Section 3 discusses some additional systems issues for our solution. Section 4 quantitatively evaluates the performance of our algorithm using a case study of keyword index placement in distributed full-text search engines. We conclude in Section 5 with a summary of results.

1.1 Motivating Applications

Full-text search engines host inverted indices to support fast search. An inverted index is a data structure that associates each keyword with those documents that contain it, as well as other auxiliary information such as keyword occurrence locations, document digests, etc. In large-scale distributed search systems, search indices are partitioned and placed over multiple nodes[1]. When answering a k-keyword search query, typically the inverted indices of all k keywords need to be accessed. A primary performance concern is the network communication overhead between distributed indices for keywords that appear in the same queries.

Similarly, multi-object operations arise in many distributed data-driven applications. For instance, a large biological sequence database may be partitioned and placed on multiple machines for scalability. A query may search specific parts of the database depending on user specification and search results from all relevant parts are finally aggregated in a union-like fashion.

The benefit and practicality of our study on correlation-aware object placement depends on two premises. First, the distribution of object correlations must be *skewed* enough for potentially large benefit of correlation-aware object placement. Second, the distribution must be *stable* enough so that a placement scheme customized to a particular object correlation distribution can remain effective for a significantly long time period. We believe such skewness and stability hold for many large-scale applications over long-term deployment. In particular, we show the correlations of keywords in search queries using a trace of around 29 million queries from the Ask.com search engine [1]. This trace contains a fraction of Ask.com queries received in a two-month period (January and February 2006). Trace analysis results in Figure 2 demonstrate that the distribution of keyword correlations is highly skewed in multi-keyword queries and such distribution remains stable across month-long periods.

2 Algorithmic Foundation

In this section, we present our algorithmic contribution to optimize correlation-aware object placement for multi-object operations.

2.1 Problem Definition and NP-Hardness

We formulate the correlation-aware object placement problem as follows. Given two sets, T (data objects) and N (host nodes), an object size function s, a node capacity function c, an object pair correlation function r, and a communication overhead function w, the problem is to find an object placement scheme $f : T \rightarrow N$ that satisfies Figure 3.

$$\text{minimize} \sum_{i,j \in T \wedge f(i) \neq f(j)} r(i,j) \cdot w(i,j) \qquad (1)$$

subject to the constraint:

$$\forall k \in N : \sum_{i \in T \wedge f(i) = k} s(i) \leq c(k) \qquad (2)$$

Figure 3. Original problem definition.

Specifically, $s(i)$ is the size of object i. $c(k)$ is the space capacity at node k. The object pair correlation $r(i,j)$ represents the probability for objects i and j to be requested together in any given user-level operation. For objects i and j that are placed on different nodes, $w(i,j)$ represents the

[1] Search index partitioning can be either keyword-based or document-based [14]. In keyword-based partitioning, each node hosts the inverted indices of some keywords. In document-based partitioning, each node hosts the inverted indices (of all keywords) for some documents. Both schemes have distinctive advantages and some hybrid approaches are also employed. For the simplicity of treating each keyword index as a data object, we consider pure keyword-based partitioning systems or the keyword partitioning component of hybrid systems.

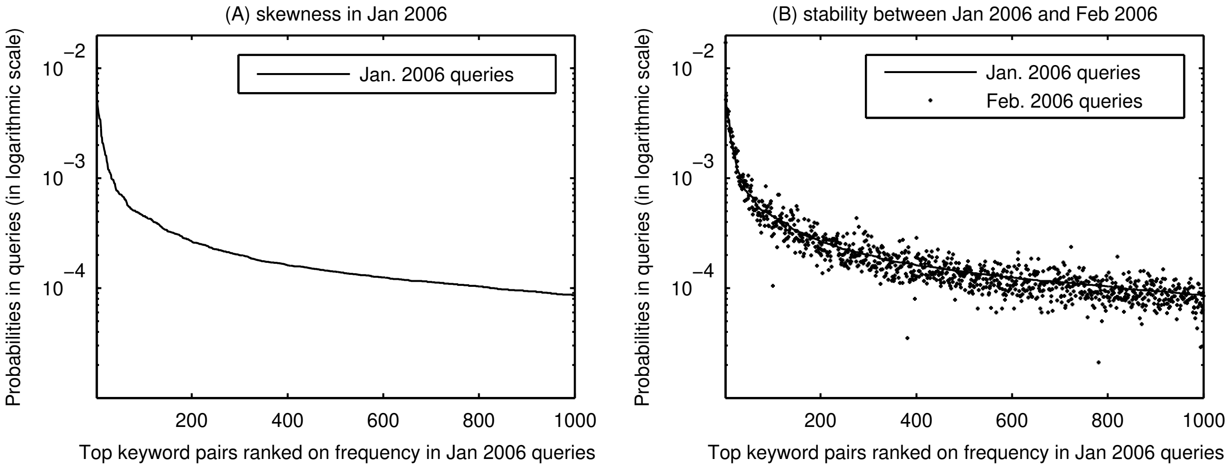

Figure 2. An illustration on the skewness and stability of keyword correlations using a trace of Ask.com [1] search queries during January and February 2006. (A) shows the skewness of keyword correlations for the most correlated 1000 keyword pairs in January 2006 queries. Note that the Y-axis is in logarithmic scale. The most correlated keyword pair is 177 times more correlated than the 1000th most correlated keyword pair. (B) shows the stability of keyword correlations between two month-long periods. The closeness between the dots (frequencies in February 2006 queries) and the curve (frequencies in January 2006 queries) indicates high stability. In particular, only 1.2% keyword pairs have correlation changes that are greater-than-twice or less-than-half the originals.

communication overhead incurred when objects i and j are requested together in an operation. To simplify our problem formulation and to allow solution with strong analytical property, we assume $w(i, j)$ is only dependent on inherent properties of objects i and j. In other words, we assume the communication cost between two specific objects i and j is deterministic. This assumption has some important implications:

- The communication cost between two objects is independent of where they are placed in the distributed system. This assumption largely holds for local-area distributed environments in which the communication latency between nodes are approximately equal. For wide-area distributed environments, our assumption holds when the optimization goal is to minimize the combined network communication volume over all links regardless of the network distance at each link.

- The communication cost between a pair of objects is independent of other requested objects in an operation requesting more than two objects. This is a strong assumption which often does not hold in practice. In these cases our analytical result only applies to two-object operations. However, we will show later in Section 3.2 how the cost of some more-than-two-object operation can be approximated using the cost of one or more two-object operations.

Given s, c, r, and w, the optimization goal of (1) minimizes the total communication cost between indices on different nodes while the constraint of (2) ensures that the total data size at an arbitrary node k does not exceed the node space capacity. We call our problem *Capacity-Constrained Assignment* (or *CCA*).

Theorem 1 *The CCA problem is NP-hard.*

Proof: Let us consider a special case of the CCA problem. Suppose there are t objects and n nodes with $t > n \geq 3$ and all nodes have an equal capacity of c. Suppose the first n objects have equal size of $\frac{c}{2} < s < c$, which means that they must be bijectively placed on the n nodes. Otherwise the per-node capacity limit would be exceeded. We also assume that the total size of all other objects is no more than $c - s$, which means that these objects can be freely placed at any node without violating the capacity constraint.

Now the problem becomes a minimum n-way cut problem with t vertexes, n out of which are pre-determined terminals. Furthermore, by carefully choosing t, n, object sizes, and node capacities, the above restriction can cover every possible instance of minimum n-way cut problems. Consequently, given the known NP-hardness of the minimum n-way cut problem ($n \geq 3$), we know that the CCA problem is NP-hard since one of its special cases is NP-hard. ■

2.2 A Polynomial-Time, Randomized Solution

We derive a polynomial-time, randomized solution for the CCA problem, by representing it as an integer programming problem, relaxing it to linear programming, solving the relaxation in polynomial time, and finally rounding the solution to integers via a randomized approach. At the high level, this multi-step process of approximating NP-hard problems has been used previously [4, 9].

For all $i, j \in T$, $k \in N$: $r(i, j)$, $w(i, j)$, $s(i)$, and $c(k)$ as we defined earlier are all known. Let $S = \sum_{i \in T} s(i)$. The CCA problem as defined in Figure 3 is equivalent to the linear integer programming problem shown in Figure 4. In this problem definition, the constraints in (4–8) ensure that (3) minimizes the total inter-node communication overhead while (9) maintains the per-node capacity constraint.

The solution to the above linear integer programming problem provides the optimal object placement for our formulated problem. However, it is generally difficult to find a solution since integer programming problems are known to be NP-hard. To address this, we relax the constraint in (4) to $x_{i,k} \geq 0$, which transforms the problem into a linear programming problem. The solution to this new program is polynomial-time computable and it always satisfies $0 \leq x_{i,k} \leq 1$. This can be viewed as a fractional object placement scheme (where an object can be split into arbitrary parts and placed at different nodes) and needs to be rounded to an integer solution, with the hope that the rounding process does not amplify the objective function too much.

It is obvious that the simple nearest integer rounding does not work here since it may generate all zeros and violate (5). An appropriate rounding scheme should choose exactly one node $k \in N$ for each object $i \in T$, with probability $x_{i,k}$. However, doing so naively without considering correlations between different objects is insufficient since it may incur considerable increase in the optimization target (total communication overhead). For instance, two objects i, j with identical fractional node placement probabilities (hence have zero contribution to the optimization goal in (3)) may be placed at different nodes, which causes unbounded increase on the optimization target. To address this, we use the randomized rounding technique in [4] that places object i at node k with probability $x_{i,k}$, while maintaining the the optimal expected target value. The rounding approach is illustrated in Algorithm 2.1.

$$\text{minimize} \sum_{i,j \in T} r(i,j) \cdot w(i,j) \cdot z_{i,j} \qquad (3)$$

subject to constraints:

$$\forall i \in T, \forall k \in N: \quad x_{i,k} \in \{0, 1\} \qquad (4)$$

$$\forall i \in T: \quad \sum_{k \in N} x_{i,k} = 1 \qquad (5)$$

Comment: $x_{i,k}$ are integer variables such that $x_{i,k} = 1$ when object i is placed at node k and $x_{i,k} = 0$ otherwise. (5) enforces that each object is placed at exactly one node.

$$\forall i, j \in T, r(i,j) > 0, \forall k \in N: \quad y_{i,j,k} \geq x_{i,k} - x_{j,k} \qquad (6)$$

$$\forall i, j \in T, r(i,j) > 0, \forall k \in N: \quad y_{i,j,k} \geq x_{j,k} - x_{i,k} \qquad (7)$$

Comment: The intermediate variable $y_{i,j,k}$ is enforced to be equal to $\max(x_{i,k} - x_{j,k}, x_{j,k} - x_{i,k}) = |x_{i,k} - x_{j,k}|$.

$$\forall i, j \in T, r(i,j) > 0: \quad z_{i,j} = \frac{1}{2} \cdot \sum_{k \in N} y_{i,j,k} \qquad (8)$$

Comment: The intermediate variable $z_{i,j} = 0$ if objects i, j are placed on the same node and $z_{i,j} = 1$ if otherwise.

$$\forall k \in N: \quad \sum_{i \in T} x_{i,k} \cdot s(i) \leq c(k)$$

$$\Leftrightarrow \forall k \in N: \quad \sum_{i \in T, k' \neq k} x_{i,k'} \cdot s(i) \geq S - c(k) \qquad (9)$$

Comment: Capacity constraint.

Figure 4. Linear integer programming problem definition.

2.3 The Optimality of the Expected Target Value

Here we will show that the expected optimization target value of the rounded solution is equal to the optimal target value of the relaxed linear programming problem.

Lemma 1 *After the rounding process, object i is placed at node k with probability $x_{i,k}$.*

Proof: In each step, the probability that object i is placed at node k is proportional to $x_{i,k}$. Furthermore, i must be placed in some step of the rounding process. Hence after the whole process, i goes to k with probability $x_{i,k}$ no matter in which step i is placed. ■

Lemma 2 *For arbitrary objects i, j, the probability that i, j*

Algorithm 2.1: ROUNDING($\{x_{i,k}\}$)

Input: $\{x_{i,k} \mid 0 \leq x_{i,k} \leq 1, i \in T, k \in N\}$, the fractional solution to be rounded.
Returns: $\{\bar{x}_{i,k} \mid \bar{x}_{i,k} \in \{0,1\}, i \in T, k \in N\}$, the rounded solution.

Initialize $\{\bar{x}_{i,k} \mid i \in T, k \in N\}$ as all zeros.
while there still exists not-yet-placed objects
do
  $r \leftarrow$ a random number in $[0,1]$
  $k \leftarrow$ a node chosen randomly from N

  /* For each not-yet-placed object i, place i at node k with probability $x_{i,k}$ */
  for each not-yet-placed $i \in T$
    do
      if $r \leq x_{i,k}$
      then $\bar{x}_{i,k} \leftarrow 1$

return ($\{\bar{x}_{i,k}\}$)

---

are placed at different nodes by the rounding process is at most $z_{i,j}$.

Proof: i, j are placed at different nodes only if they are not placed in the same step. If object i is placed first, then we have

$$
\begin{aligned}
&Prob(j \text{ is not placed when } i \text{ is placed}) \\
&= \sum_{k \in N} Prob(i \text{ goes to node } k \text{ in a step} \\
&\qquad\qquad \wedge\ j \text{ is not placed in that step})
\end{aligned} \tag{10}
$$

where $Prob(i$ goes to node k in a step$)$ is $x_{i,k}$ according to Lemma 1. The conditional probability:

$$
\begin{aligned}
&Prob(j \text{ is not placed in that step} \mid i \text{ goes to node } k \text{ in a step}) \\
&\leq \frac{\max(x_{i,k} - x_{j,k}, 0)}{x_{i,k}}
\end{aligned} \tag{11}
$$

Hence (10) is upper-bounded by:

$$
\sum_{k \in N} x_{i,k} \cdot \frac{\max(x_{i,k} - x_{j,k}, 0)}{x_{i,k}} = \sum_{k \in N} \max(x_{i,k} - x_{j,k}, 0) \tag{12}
$$

Similarly, if object j is placed first, then

$$
\begin{aligned}
&Prob(i \text{ is not placed when } j \text{ is placed}) \\
&= \sum_{k \in N} Prob(j \text{ goes to node } k \text{ in a step} \\
&\qquad\qquad \wedge\ i \text{ is not placed in that step}) \\
&\leq \sum_{k \in N} \max(x_{j,k} - x_{i,k}, 0)
\end{aligned} \tag{13}
$$

Given that $\sum_{k \in N} x_{i,k} = \sum_{k \in N} x_{j,k} = 1$ and $\forall k \in N, 0 \leq x_{i,k}, x_{j,k} \leq 1$, we have $\sum_{k \in N} \max(x_{i,k} - x_{j,k}, 0) = \sum_{k \in N} \max(x_{j,k} - x_{i,k}, 0)$. In addition, $\sum_{k \in N} \max(x_{i,k} - x_{j,k}, 0) + \sum_{k \in N} \max(x_{j,k} - x_{i,k}, 0) = \sum_{k \in N} |x_{i,k} - x_{j,k}| = 2z_{i,j}$. Hence $\sum_{k \in N} \max(x_{i,k} - x_{j,k}, 0) = \sum_{k \in N} \max(x_{j,k} - x_{i,k}, 0) = z_{i,j}$. Consequently, no matter either i or j is placed first, the probability that i, j go to different nodes is upper-bounded by $z_{i,j}$. ∎

Theorem 2 *For the rounded solution generated by Algorithm 2.1, its expected value of the total communication cost defined in (3) is identical to that of the optimal solution to the CCA problem defined in (3–9).*

Proof: According to Lemma 2, the rounding algorithm maintains the expected minimization object value of the fractional solution to the relaxed linear programming problem ($x_{i,k} \in \{0,1\}$ relaxed to $0 \leq x_{i,k} \leq 1$). Furthermore, the optimal solution to the relaxed linear programming problem obviously should not incur higher communication cost than the optimal solution to the original linear integer programming problem defined in (3–9). ∎

Theorem 2 shows that our algorithm generates a randomized object placement scheme whose expected total communication cost is optimal. Note that this is only a probabilistic guarantee. In other words, a particular solution generated by our randomized rounding may produce worse results. To achieve a high confidence in the performance of the produced object placement, we can repeat the randomized rounding several times and pick the best solution.

Theorem 3 *For the rounded solution generated by Algorithm 2.1, its expected value of total object size at each node does not exceed the per-node capacity.*

Proof: Before the rounding, the fractional solution ($\{x_{i,k}\}$) to the relaxed linear programming problem strictly satisfies the capacity constraint as given in (9) — $\forall k \in N, \sum_{i \in T} x_{i,k} \cdot s(i) \leq c(k)$. According to Lemma 1, the rounding process places object i at node k with probability $x_{i,k}$. Consequently, the expected value of the total object size at node k is

$$
\sum_{i \in T} x_{i,k} \cdot s(i) \leq c(k)
$$

∎

Note that Theorem 3 does not indicate a strict adherence to the per-node capacity constraint. Instead, it only ensures that the expected per-node load is below the capacity. To address this in practice, conservative capacities may be used for the purpose that a group of placed objects with aggregate size slightly larger than its node capacity will be tolerated.

2.4 Related Work

The optimization problem studied in this paper is a variant of the classical Quadratic Assignment (QA) problem [5], which is known to be NP-hard [10] and no polynomial-time solution with a constant approximation ratio is known. Given n facilities and n locations with distance specified for each pair of locations and the amount of commodity flow given for each pair of facilities, the QA problem is to place all facilities to different locations in order to minimize the total flow cost — the sum of the distances multiplied by the corresponding flows.

The QA problem can be tailored to model our optimization problem, where facilities, locations, and supply flows become data objects, nodes, and inter-node communications in our studied scenario. On the one hand, our problem is more complex in that the number of data objects and the number of nodes may not be equal. Multiple objects at one node is allowed in our case as long as the node capacity permits it. On the other hand, we make a simplification that the varying distances between different node pairs do not play a role in the optimization target — the overall communication cost. This simplification allows us to develop an randomized approximation solution with its expected target value guaranteed to be optimal.

Our studied problem also has close connections with other NP-hard problems, including minimum k-way cut ($k \geq 3$) and balanced uniform metric labeling. The minimum k-way cut problem can be viewed as a simplified case of our problem in which there are k nodes (terminals), each with one pre-placed object, and the problem is to find an optimal placement scheme for other not-yet-placed objects. The best known approximation ratio for the minimum k-way cut problem is $1.5 - \frac{1}{k}$ [2]. The balanced uniform metric labeling problem [9] is a class of metric labeling problems [4] with uniform distance metrics and label capacities, i.e., there is an upper-limit on how many objects can receive a label. The best known polynomial-time solution has an approximation ratio of $O(\log n)$ (n is the number of nodes) while maintaining bounded number of objects placed to each label [9]. In comparison, we seek a randomized solution that optimizes the expected target value, rather than improving the approximation ratio of exact solutions.

Our problem is also similar to a variant of the task assignment problem [6, 7, 12], which allocates a number of tasks to a distributed system of heterogeneous processors such that the total execution and communication cost are minimized while per-processor constraints are satisfied. Task assignment problems typically have heuristic solutions that focus on online efficiency for small problem sizes. Consequently, they are not directly applicable to our problem, which seeks the offline optimal assignment scheme for a large number of data objects.

3 Additional Systems Issues

3.1 Offline Computation Overhead

We examine the offline computation overhead for the linear programming problem under realistic problem sizes. Let T denote the set of objects. Let N denote the set of nodes. Let $E = \{(i,j) | i,j \in T \wedge r(i,j) > 0\}$, where $r(.,.)$ indicates the object pair correlation. Let us consider the linear program defined in (3–9) with $x_{i,k}$ relaxed to $x_{i,k} > 0$. The total number of variables ($x_{i,k}$'s, $y_{i,j,k}$'s, and $z_{i,j}$'s) is obviously

$$|T| \cdot |N| + |E| \cdot |N| + |E|.$$

We believe that in many large-scale data-intensive applications, there are often only a few other objects with non-trivial correlation with a given object. In other words, E is a sparse set over $T \times T$ with $E = O(|T|)$. Therefore the total number of variables can be viewed as $O(|T| \cdot |N|)$. Based on the same assumption, the number of program constraints in (4–9) is

$$|T| \cdot |N| + |T| + 2|N| \cdot |E| + |E| + |N| = O(|T| \cdot |N|)$$

In summary, the number of variables and constraints used in our linear program is $O(|T| \cdot |N|)$.

Even with the assumption of $E = O(|T|)$, there may still be many variables and constraints in the linear program for large-scale applications. This may require a significant amount of offline computation time to generate the optimal object placement scheme on current linear programming software. Fortunately, due to the high skewness of object popularity in many applications (e.g., Zipf-like popularity distribution for web objects), we may not have to deal with all objects in the dataset. By limiting the scope of placement optimization on a small number of important objects (dominant in access frequency and/or object size) and using random placement for others, we may trade communication overhead savings for less offline computation. We call this *important-object partial optimization*. In Section 4, we will quantitatively evaluate the effectiveness of our proposed object placement strategy with varying optimization scopes.

3.2 More-Than-Two-Object Operations

Our solution derived in Section 2 assumes that the communication cost between a pair of objects is independent of other requested objects in an operation requesting more than two objects. This is a strong assumption which may not hold in practice. Therefore our analytical result effectively only applies to two-object operations. However, with more knowledge on the nature of multi-object operations and how they are performed, we may approximate the cost of some more-than-two-object operation using the cost of one or more two-object operations.

Some multi-object operations perform intersection-like aggregation over multiple sets. For example in full-text keyword search engines, inverted indices answer a multi-keyword query by returning the ranked document list of the intersection of all keywords' corresponding indices. Similarly, a database join query intersects row sets of multiple tables. Intersection-like operations typically process two smallest objects first and their intersection result tends to become so small that further intersections require little communication. Under such schemes, we can approximate the communication cost of a multi-object operation as that of an operation that requests only the two smallest objects. Correspondingly, we adjust our definition of object pair correlation to be the probability that they are the two smallest objects requested in any given operation.

Some other multi-object operations perform union-like aggregation over multiple objects. An example is the union of search results from multiple datasets. In a simplistic fashion, we transfer all objects to the node at which the largest object is located and then perform the union locally. In this case, we can approximate the communication cost of a multi-object operation using the aggregate cost of a sequence of two-object operations, each of which involves the largest object and each other requested object in the operation.

3.3 Other Node Capacity Constraints

In addition to the storage capacity constraint explicitly considered in our problem definition, other node capacity constraints such as network bandwidth and CPU processing capability may also be present. In principle, we can address these problems by introducing more capacity constraints into our linear programing problem in a way similar to (9), e.g., ensuring that the total resource usage at each node does not exceed the per-node bandwidth limit or CPU processing capability. Thus the general framework of our solution should still be applicable. A quantitative consideration on these additional capacity constraints falls beyond the scope of this paper.

4 A Quantitative Case Study

We experimentally evaluate our proposed correlation-aware object placement using a case study on distributed keyword index placement for full-text search engines. Our experimental study is driven by real-world web document datasets and query traces. More specifically, we evaluate the feasibility of important-object partial optimization to achieve manageable offline computation cost (Section 4.2). We then provide evaluation results on the effectiveness of our proposed algorithm in reducing communication overhead (Section 4.3).

4.1 Evaluation Setup

Our evaluation dataset contains 3.7 million web pages and 6.8 million web queries. The web pages are crawled based on URL listings of the Open Directory Project [3]. The queries are from a partial query log at the Ask.com search engine [1] over the week of January 6–12, 2002 and there are an average of 2.54 keywords per query. The complete vocabulary consists of the 253,334 words that appear in our query log. The web pages are preprocessed by removing HTML tags and trivially popular words using the stopword list of the SMART software package [11]. After pre-processing, the average number of distinct words per page is approximately 114.

In our implemented inverted indices, each item of an inverted index contains an 8-byte page ID (the MD5 digest of the corresponding page URL). Note that inverted indices in real search engines usually contain other information such as keyword frequencies, keyword occurrence locations, and page digests. We omit them here because these information only help ranking the search results, which is not the focus of this study. The index size of each keyword can be easily computed based on our implemented inverted indices. With the knowledge of keyword index sizes and the query log, we can estimate the inter-keyword communication overhead of inverted index intersection.

We implement a prototype search engine to evaluate the communication cost saving of our proposed correlation-aware object placement. Driven by the query log, the prototype locates the nodes that contain the inverted indices of the queried keywords, performs intersection operations to generate search results, and logs the communication overhead incurred during this process for comparison purposes. We do not consider the communication cost of returning the final ranked search results because the sizes of these communications are relatively small and their exact sizes are determined by other factors independent of the index placement.

We consider three strategies for placing keyword indices at nodes.

- *Random hash-based index placement*. In this strategy, the inverted index of each keyword is placed at a node based on its MD5 hash code. More specifically, we divide the hash code by the number of nodes and use the remainder as the ID of the placed node. Random hash-based index placement or its variants are commonly employed in practice today.

- *Correlation-aware index placement using linear programming with randomized rounding (or LPRR)*. This strategy follows our proposed algorithm in Section 2. Due to potentially large cost of linear programming computation, we consider a variant of this scheme that only optimizes a small subset of most important keywords (as explained in Section 3.1). The remaining keyword indices will be placed using random hashing. For each more-than-two-keyword search query, we approximate its cost using a two-keyword search query on the two keywords with smallest inverted indices (as explained in Section 3.2). An important concern for LPRR is that the communication reduction must be achieved by a balanced placement, without causing excessively above-average load at particular nodes. We achieve this by setting a per-node index size capacity constraint in our linear programming optimization. Specifically, our constraint is set at two times the average per-node index size, i.e., no more than twice the average per-node load is allowed at each node.

- *Correlation-aware index placement using a greedy heuristic*. This strategy places indices by considering keyword correlations in a greedy way. Specifically, we examine keyword pairs in the descending order of their query correlations and always place the most correlated pair on the same node as long as the node capacity permits it. The greedy approach serves as a comparison basis that uses simple correlation-aware heuristics.

Note that when correlation-aware index placement is used, each node should maintain a look-up table of keyword index locations. The storage consumption and look-up time of such tables are not a performance concern when the keyword vocabulary size is moderate (253,334 on our dataset). Further, the proposed limited-scope partial optimization also reduces overhead associated with this look-up table since the table only needs to contain those important keywords within the optimization scope.

4.2 Feasibility of Important-Object Partial Optimization

In the ideal case, our correlation-aware index placement should consider all keywords. However, as explained in Section 3.1, this would result in tens of millions variables and constraints in the linear programming calculation. Fortunately, real-life datasets exhibit high skewness so that only a very small number of keywords may cover the vast majority of the inter-keyword communication cost[2] (the optimization target of our linear programming) and keyword index size (the capacity constraint of our linear programming). By only considering these important keywords, the

[2]Here the communication cost between keyword i and j is $r(i,j) \cdot w(i,j)$, as defined in Section 2.1.

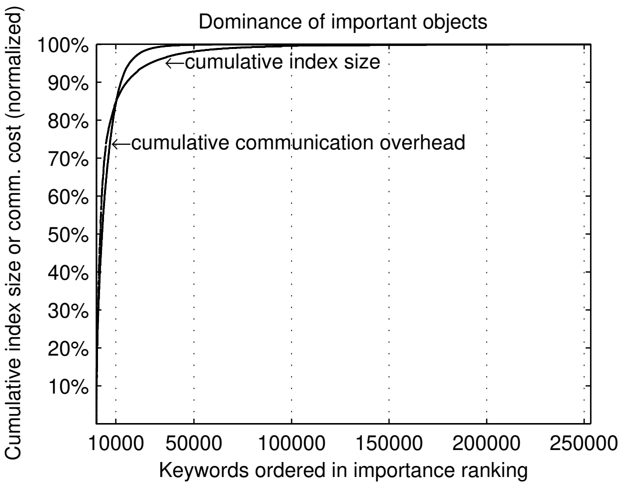

Figure 5. The dominance of most important keywords in cumulative keyword index sizes and inter-keyword communication cost.

offline computation cost for the linear programming optimization would not be too excessive.

We assess such object skewness using a simple *keyword importance ranking* scheme. In this scheme, we first rank all keyword pairs according to their inter-keyword communication cost. The keyword appearance order in this keyword pair ranking is then considered as our keyword importance ranking. Those keywords that do not involve any inter-keyword communications (e.g., those never queried together with other keywords) are ranked last.

Figure 5 shows that a few most important keywords both cover a large proportion of total communication overhead and account for a large proportion of the total index size. This justifies the feasibility of important-object partial optimization, which limits the optimization scope to a small number of most important keywords and applies random placement for other keywords. Compared to the full optimization, such a strategy is expected to achieve diminished yet still significant communication savings. Furthermore, the placement strategy generated by partial optimization tends to also maintain space constraints as keywords with large index sizes are mostly considered in the placement.

We use LPsolve [8], a popular linear programming software, to implement our algorithm and compute object placement schemes for the important keywords. The computation takes no more than 48 hours when we consider up to 10000 most important keywords — a manageable offline computation cost for large-scale search systems. For the remaining keywords, we simply determine their host nodes using random hash-based placement.

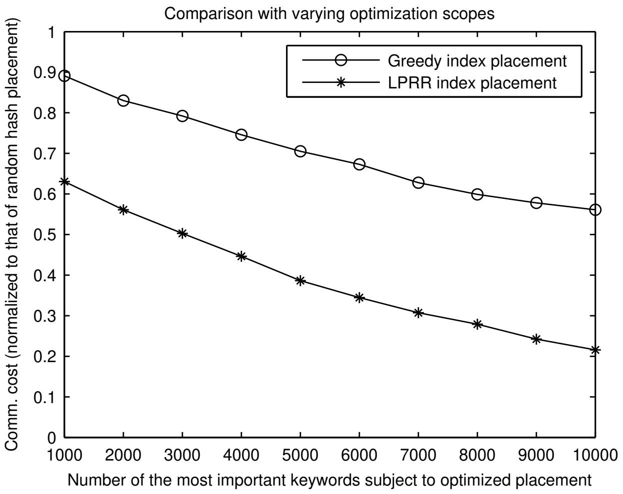

Figure 6. Communication overhead reduction of correlation-aware object placements. Results are with a range of varying optimization scopes (the number of most important keywords that are subject to correlation-aware index placement). Those keywords not in the optimization scope are placed using random hash-based node assignment. The distributed system consists of 10 nodes.

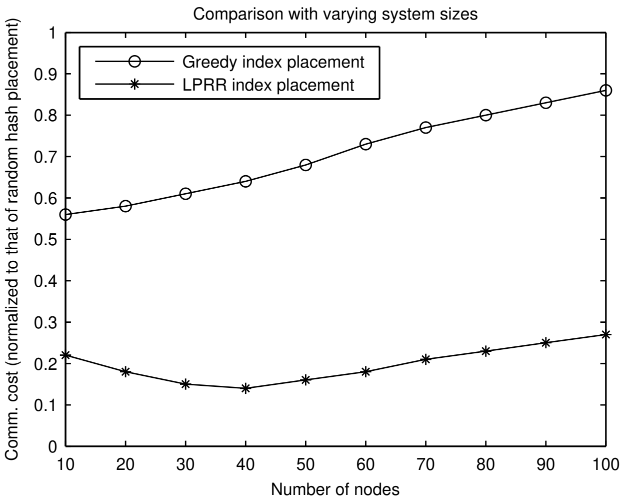

Figure 7. Communication overhead reduction of correlation-aware object placements. Results are with a range of different system sizes. The optimization scope is set at the most important 10000 keywords.

4.3 Communication Overhead Reduction

We compare the communication overhead of our LPRR correlation-aware index placement scheme with those of greedy index placement and random hash-based placement. Our evaluation is performed at a variety of optimization scopes (1000–10000 most important keywords subject to optimization) and at a range of different system sizes (10–100 nodes).

Results with varying optimization scopes Figure 6 shows that a significant amount of communication saving can be achieved even when only a small number of most important keywords are subject to correlation-aware index placement. For example, 78% communication saving can be realized when only the most important 10000 keywords are subject to LPRR correlation-aware placement. Furthermore, compared with our proposed correlation-aware placement strategy, the simple correlation-aware heuristic can achieve diminished yet still significant communication reduction (up to 44% savings)

Results with varying system sizes Figure 7 shows that our LPRR index placement scheme achieves 73–86% communication reduction compared to random hash placement. Specifically, when the system size is small (less than 40 nodes), the communication savings increase as the number of nodes grows. This is because the keywords can be well clustered into a small number of co-placed groups (with low inter-group communication) by using our optimization approach. Moreover, under such a small system size, increasing the number of nodes significantly increase the communication cost of random index placement — on n nodes, two keywords lie on different nodes with probability $\frac{n-1}{n}$, which increases quickly when n is small. When the system size becomes large (at or over 50 nodes), the communication reduction of our LPRR object placement scheme diminishes as the number of nodes increases. This is because our optimization is less effective in achieving low communication clustering on a large number of nodes while random index placement receives little effect from the number of nodes when n is large. We also note that greedy index placement is effective only when the number of nodes is small — when per-node space capacity is large. This is because greedy index placement can only discover tightly correlated keyword groups with large group sizes. It is more likely to get trapped into a local optimum when pursuing a finer grouping granularity (more nodes).

5 Summary

Multi-object operations arise in many distributed data-intensive applications. The object placement affects the performance of these operations due to communication overhead required when requested objects are placed on different nodes. In this paper, we identify the problem of finding optimal correlation-aware object placement for multi-object operations. We represent the problem as a network resource allocation problem that seeks a balanced allocation scheme with minimum communication overhead. We show that the problem is NP-hard and propose a polynomial-time, randomized solution that optimizes the *expected* value of the optimization target. More specifically, we approximate an integer linear programming problem using a relaxed regular linear programming problem followed by a randomized rounding of the solutions.

We experimentally evaluate the effectiveness and feasibility of our scheme using a trace-driven case study on distributed keyword index placement for full-text search engines. Taking advantage of a high skewness in the dataset, we can limit the scope of our optimized placement only to a small number of most important keywords. This will help realize a moderate offline computation cost for generating the placement strategy. Over a variety of optimization scopes and system sizes, our proposed correlation-aware placement achieves 37–86% communication reduction over the baseline random hash-based placement and 30–78% reduction over a simple heuristic of greedy correlation-aware placement.

This work is related to our other studies of using known object popularity information to customize object replication degrees [15] and Bloom filter object hash counts [13]. Together, our results demonstrate substantial system adaptation benefits by exploiting skewed but stable data access distributions in real-world data-intensive applications.

Acknowledgments

We thank Lingkun Chu and Tao Yang at Ask.com for providing us the keyword query traces used in this study. We would also like to thank the anonymous ICDCS reviewers for their valuable comments that helped improve this paper.

References

[1] Ask.com Search (formerly Ask Jeeves). http://www.ask.com.

[2] G. Calinescu, H. Karloff, and Y. Rabani. An Improved Approximation Algorithm for Multiway Cut. In *30th ACM Symp. on Theory of Computing*, pages 48–52, 1998.

[3] The Open Directory Project. http://www.dmoz.com.

[4] J. Kleinberg and E. Tardos. Approximation Algorithms for Classification Problems with Pairwise Relationships: Metric Labeling and Markov Random Fields. In *40th IEEE Symp. on Foundations of Computer Science*, pages 14–23, 1999.

[5] T. Koopmans and M. Beckman. Assignment Problems and the Location of Economics Activities. *Econometrica*, 25:53–76, 1957.

[6] C. Lee and K. Shin. Optimal Task Assignment in Homogeneous Networks. *IEEE Trans. on Parallel and Distributed Systems*, 8(2):119–128, 1997.

[7] V. Lo. Heuristic Algorithms for Task Assignment in Distributed Systems. *IEEE Trans. on Computers*, 37(11):1384–1397, 1988.

[8] LPsolve. http://lpsolve.sourceforge.net/5.1/.

[9] J. Naor and R. Schwartz. Balanced Metric Labeling. In *37th ACM Symp. on Theory of Computing*, pages 582–591, 2005.

[10] S. Sahni and T. Gonzalez. P-Complete Approximation Problems. *Journal of ACM*, 23:555–565, 1976.

[11] SMART. ftp://ftp.cs.cornell.edu/pub/smart.

[12] B. Ucar, C. Aykanat, K. Kaya, and M. Ikinci. Task Assignment in Heterogeneous Computing Systems. *Journal of Parallel and Distributed Computing*, 66(1):32–46, 2006.

[13] M. Zhong, P. Lu, K. Shen, and J. Seiferas. Optimizing Data Popularity Conscious Bloom Filters. In *27th ACM Symp. on Principles of Distributed Computing*, 2008.

[14] M. Zhong, J. Moore, K. Shen, and A. Murphy. An Evaluation and Comparison of Current Peer-to-Peer Full-Text Keyword Search Techniques. In *8th Int'l Workshop on the Web and Databases*, pages 61–66, 2005.

[15] M. Zhong, K. Shen, and J. Seiferas. Replication Degree Customization for High Availability. In *Third EuroSys Conf.*, pages 55–68, 2008.