DEWS2003 2-B-01

Effective load-balancing of peer-to-peer systems

Anirban Mondal Kazuo Goda Masaru Kitsuregawa

Institute of Industrial Science
University of Tokyo, 4-6-1, Komaba, Meguro-ku,
Tokyo 153-8505, Japan
{anirban,kgoda,kitsure}@tkl.iis.u-tokyo.ac.jp

ABSTRACT

The growing popularity of peer-to-peer (P2P) systems has necessitated the need for managing huge volumes of data efficiently to ensure acceptable user response times. Dynamically changing popularities of data items and skewed user query patterns in P2P systems may cause some of the peers to become bottlenecks, thereby resulting in severe load imbalance and consequently increased user response times. An effective load-balancing mechanism becomes a necessity in such cases. Such load-balancing can be achieved by efficient *online* data migration/replication. While much work has been done to harness the huge computing resources of P2P systems for high-performance computing and scientific applications, issues concerning load-balancing with a view towards faster access to data for normal users have *not* received adequate attention. Notably, the sheer size of P2P networks and the inherent dynamism of the environment pose significant challenges to load-balancing. The main contributions of our proposal are three-fold. First, we view a P2P system as comprising clusters of peers and present techniques for both intra-cluster and inter-cluster load-balancing. Second, we analyze the trade-offs between the options of migration and replication and formulate a strategy based on which the system decides at run-time which option to use. Third, we propose an effective strategy aimed towards automatic self-evolving clusters of peers. Our performance evaluation demonstrates that our proposed technique for inter-cluster load-balancing is indeed effective in improving the system performance significantly. To our knowledge, this work is one of the earliest attempts at addressing load-balancing via both *online* data migration and replication in P2P environments.

Keywords: Peer-to-peer systems, load-balancing, data migration, data replication.

1. INTRODUCTION

The emergence of larger, faster and more powerful computer networks, which have the capability to connect thousands of computers worldwide, has opened new as well as exciting avenues for research in peer-to-peer (P2P) computing. P2P systems [4, 5, 6] typically comprise a tremendously large number of geographically distributed and distributively owned computers and the objective of such systems is to facilitate effective sharing of resources across the entire P2P network.

While much work has been done to harness the huge computing resources of P2P systems for high-performance computing and scientific applications, issues concerning load-balancing with a view towards faster access to data for normal users have *not* received adequate attention. Given the unprecedented growth of data in existing P2P systems, efficient data management has become a necessity to provide acceptable response times to user requests. Notably, a dependable[1] and load-balanced P2P system not only provides considerable savings in network bandwidth, but also ensures that the benefit of powerful networks extends well beyond e-commerce and scientific applications to individuals living in this world. This is a revolution in itself and underlines the novelty of a dependable P2P system.

In P2P systems, huge volumes of data are typically declustered across a large number of peers. Incidentally, since every user has the freedom to share *any data* with other users, it is *not* possible to control the initial data distribution. Moreover, given the inherent dynamism of the P2P environment, *no* static data distribution can be expected to guarantee good load-balancing. Changing popularities of various data items and skewed query patterns may result in disparate number of disk accesses at some of the *hot* peers, thereby causing severe load imbalance throughout the system. The hot peers become bottlenecks and this leads to increased user response times, thus degrading system performance significantly. Hence a load-balancing mechanism becomes a necessity in such cases and such load-balancing can be achieved by efficient *online* data migration/replication.

This paper proposes a dependable and load-balanced P2P system. The main contributions of our proposal are three-fold.

1. We view a P2P system as comprising clusters of

[1]By *dependability*, we mean high data availability.

peers and present techniques for both intra-cluster and inter-cluster load-balancing. Notably, load-balancing facilitates reduced query response times.

2. We analyze the trade-offs between the options of migration and replication and formulate a strategy based on which the system decides at run-time which option to use. Incidentally, analysis of trade-offs between migration and replication is expected to facilitate load-balancing.

3. We propose an effective strategy aimed towards automatic self-evolving clusters of peers. This is important since P2P environments are inherently dynamic.

The remainder of this paper is organized as follows. Section 2 discusses related work, while Section 3 presents an overview of our proposed system. The proposed load-balancing strategy via migration/replication is presented in Section 4. Section 5 reports our performance evaluation. Finally, we conclude in Section 6.

2. RELATED WORK

Existing P2P systems such as Pastry [8] and Chord [10] emphasize specifically on query routing, while the work in [3] proposes 3 types of routing indices (RIs), namely compound RIs, hop-count RIs and exponential RIs to facilitate search in P2P systems. In particular, peers forward queries to their neighbours based on their own RIs. As such these works do *not* specifically address load-balancing.

Static load-balancing approaches [2] typically attempt to perform an intelligent initial declustering of data. Since static approaches are not adequate to deal with dynamically changing user access patterns, several dynamic load-balancing techniques [1, 11, 12] have been proposed. Notably, among these works, only the proposals in [1, 11] focus specifically on scalability issues. Incidentally, these works are not specifically aimed towards P2P systems.

The proposal in [4] discusses the *flocking* mechanism in the Condor system by means of which jobs submitted in one Condor pool may access resources belonging to another Condor pool. Gateway machines, one in each pool, act as resource brokers between the two pools and also coordinate job transfers. Note that the approach in [4] uses job migration, while we use data migration/replication. Moreover, a collaborative web mirroring system (Backslash [9]), which is run by a set of websites, has been proposed with a view towards ensuring protection from flash crowds. Notably, the work in [9] deals with load-balancing *only* in the context of a static architecture. Moreover, the assumption in [9] concerning global knowledge makes it inappropriate for P2P systems.

3. SYSTEM OVERVIEW

This section discusses an overview of the proposed system. At the very outset, we define *distance* between two clusters as the communication time τ between the cluster leaders and if τ is less than a pre-specified threshold, the clusters are regarded as *neighbours.*(Since cluster leaders will collaborate, their communication time is critical.) Notably, most existing works define a peer's load as the number of requests directed at that peer, the implicit assumption being that *all* requests are of equal size, but this does *not* always hold good in practice. To take varying request sizes into account, we define the load of P_i, L_{P_i}, as follows.
$L_{P_i} = D_i \times (CPU_{P_i} \div CPU_{Total})$ where D_i represents the number of Megabytes retrieved [2] at peer P_i during a given time interval T_i, CPU_{P_i} denotes the CPU power of P_i and CPU_{Total} stands for the total CPU power of the cluster in which P_i is located.

Proposed System Framework

In the interest of managing huge P2P systems effectively, we view the system as comprising several clusters, where peers are assigned to clusters such that the clusters are mutually disjoint. In our proposed strategy, every peer is assigned a unique identifier *peer_id* and *all* peers belonging to the same local area network (LAN) are initially assigned to a single cluster. Every incoming query is assigned a unique identifier *Query_id* by the peer P_i at which it arrives. *Query_id* consists of *peer_id* and *num* (a distinct integer generated by P_i). Every peer maintains its own access statistics i.e., the number of disk accesses made for each of its data items *only* during the last time interval. (Time is divided into equal pre-defined intervals at design time.) This information is used for detecting hotspots in a peer. Given the inherent dynamism of P2P environments, a moment's thought shows that *only* the most recent access statistics should be used to determine hotspots.

Each cluster is randomly assigned a leader. The job of the cluster leaders is to coordinate the activities (e.g., load-balancing, searching) of the peers in their clusters. Each cluster leader also maintains information concerning the set of categories stored both in its own cluster as well as in its neighbouring clusters. Category-related update information is periodically exchanged between neighbouring cluster leaders preferably by piggybacking such information with other messages. This facilitates effective pruning of the search space as it enables a cluster leader to decide quickly whether its cluster members contain the answer to a particular user query.

Any peer joining/leaving the system informs its respective cluster leader. In case a cluster leader itself decides to leave the system, it chooses one of the lightly loaded peers as the new cluster leader and sends a broadcast message to all the cluster members informing them about the new cluster leader. Moreover, it also transfers the necessary cluster-related information to the new cluster leader.

Incidentally, every peer typically needs to maintain some

[2]Notably, the number of Megabytes retrieved is a direct quantification of disk I/O activity.

information (e.g., meta-data structures, node routing tables) concerning their peers in order to facilitate fast access to the data. We shall collectively refer to such information as *meta-information*. In this regard, let us now examine two possible design alternatives.

1. **Significant meta-information maintenance:** The advantages of this approach are that the search operation is expected to require less time and only the peers containing the data items will be involved in answering the query. However, a serious drawback of this strategy is that the overhead required for keeping the meta-information updated may become prohibitively high owing to the following reasons:
   - A very large number of data items may be added or updated or deleted within a very short time interval.
   - Nodes may enter or leave the system frequently, thereby introducing significant amount of redundancies to the meta-information or making some of the meta-information obsolete.
   - A data item may have been migrated or replicated several times (possibly for load-balancing).
   - The meta-information may become too large to fit in the cache, thereby making disk accesses necessary.
2. **Minimal meta-information maintenance:** This approach has the advantage of low overhead for maintenance of meta-information, but the search operation may require more time than in the former approach and peers not containing the queried items may also become involved in the search.

Notably, it is important for P2P systems to be scalable over time. The first approach is not scalable over time since the meta-information keeps growing over time and consequently, the maintenance overheads associated with the meta-information also keep increasing. In contrast, we expect the second approach to be extremely scalable over time primarily owing to its low maintenance overheads. Hence we propose that each peer should only maintain a minimal amount of meta-information. As we shall see later, our proposed techniques only require minimal amount of meta-information to be maintained by each peer.

Automatic self-evolving clusters of peers

Peers may join or leave the system at any time, thereby providing a strong motivation for an automatic self-evolving strategy for peer clustering. Interestingly, the number of peers in each cluster may differ significantly. Consequently, some clusters may have very high number of peers, while some clusters may have very few peers. Since a *single* peer may not be adequate to be the leader of a huge cluster, we propose to split such huge clusters. In such cases, the leader decides upon the number N of clusters the current cluster should be split into. The leader also selects N of the most lightly loaded peers in the cluster as the respective leaders of the new clusters and assigns peers to the new cluster in a round-robin fashion, the objective being to ensure that the new clusters have approximately equal number of peers.

If a cluster C_s has very few peers, its leader sends messages to its neighbouring cluster leaders, asking for permission to join those clusters. C_s's neighbouring cluster leaders make their decisions based on their current number of peers and inform C_s's leader about their decisions. C_s's leader compiles a list of the willing cluster leaders and selects among them the cluster C_g that is nearest to itself. In case there is more than one cluster that is approximately the same distance from C_s's leader, the one with the least number of members is selected. C_s's members join C_g. Either C_g's leader or C_s's leader becomes the leader of the combined cluster, depending upon which cluster initially had a higher number of peers. Any ties are resolved arbitrarily.

Efficient quality-oriented search with user feedback

We define a peer as *relevant* to a query Q if it contains at least a non-empty subset of the answers to Q. Moreover, we define a cluster as being *active* with respect to Q if *at least* one of its members is still processing Q.

Whenever a query Q arrives at a peer P_i, Q is assigned one or more categories by P_i and P_i becomes the initiator of Q. Moreover, P_i keeps track of the leaders of those clusters that are *active* with respect to Q. If P_i is *not* relevant to Q, it sends Q to its cluster leader C_i. If C_i determines from its category-related information that Q is *not* relevant to any of its cluster members, it forwards Q to members of set ξ. (Set ξ comprises *only* those neighbouring cluster leaders of C_i whose members store *at least* one of the categories associated with Q. In case *none* of the neighbouring clusters have at least one of Q's categories, set ξ will consist of *all* the neighbouring cluster leaders of C_i.) These cluster leaders, in turn, will try to answer Q via their cluster members and if their cluster members are *not* relevant to Q, the same process continues.

We specifically note that when any of the peers returns results to the user, the search terminates *only* if the user indicates that he/she is satisfied with the results, otherwise the search continues. If the user is satisfied with the results, P_i sends a message concerning termination of Q to those cluster leaders that are *active* with respect to Q and the *active* cluster leaders, in turn, broadcast the message in their respective clusters to ensure the termination of Q. Interestingly, our proposed search algorithm is quality-oriented. However, to prevent malicious users from degrading system performance, a time-out mechanism is used such that the query initiator will automatically send a message for termination of the query after time t_{max} has elapsed, the value of t_{max} being decided at design time. For our experiments, we have set

t_{max} to a large value of 1 hour primarily because security is *not* our primary focus in this paper. Figure 1 depicts the search algorithm, while Figure 2 shows how user satisfaction is guaranteed.

Algorithm P2Psearch( )
/* Q comes to one of the peers, say P_i */
if P_i is relevant to Q
  P_i returns the results to the user
  wait_for_user ( )
  if **terminate** message received then **end**
else
  P_i sends Q to its cluster leader C_i
  C_i decides the set of possible categories, χ, for Q
  if C_i is relevant to Q
    C_i propagates the results to the user
    wait_for_user ( )
    if **terminate** message received then **end**
  else
    C_i broadcasts Q and χ to its cluster members
    if any of the peers is relevant to Q
      for each relevant peer
       the results are propagated to the user
       wait_for_user ( )
       if **terminate** message received then **end**
    else
      while (1)
        Q and χ are sent to a set ξ of cluster leaders
        Each member of ξ sends **inform** message to P_i
        ξ's members broadcast Q in their clusters
        if *any* of the peers is *relevant*
          for each relevant peer
           the results are propagated to the user
           wait_for_user ( )
           if **terminate** message received then **break**
      endwhile
end

Figure 1: Search Algorithm

Algorithm wait_for_user ( )
The results of query Q are received by the user
if user is satisfied with the results
  user sends **terminate** message to the leaders of
  active cluster leaders
end

Figure 2: Algorithm executed by user to ensure the *quality* of search results

4. LOAD-BALANCING

This section discusses *intra-cluster* and *inter-cluster* load-balancing via migration and replication of data. While intra-cluster load-balancing refers to balancing the loads within a particular cluster, inter-cluster load-balancing attempts to ensure load-balancing among the clusters i.e., to achieve load-balancing across the system as a whole.

Migration vs Replication

Load-balancing can be achieved by transferring hot data from heavily loaded peers to lightly loaded peers via *data migration* or *data replication*. Note that unlike replication, migration implies that once hot data have been transferred to a destination peer, they will be **deleted** at the source peer. Now let us study the trade-offs between migration and replication.

If replication is used, in spite of several replicas of a specific data item D_i, a specific replica may keep getting accessed a disproportionately large number of times because the search is completely decentralized, thereby providing no absolute guarantee of load-balancing[3]. On the other hand, replication increases data availability albeit at the cost of disk space. Hence, a periodic 'cleanup' of the replicas becomes necessary since the hot data yesterday may be cold today, thereby implying that the replicas are no longer needed. Moreover, issues regarding the replication of large data items need to be examined. In essence, our aim is to ensure that replication performed for short-term benefit does not cause long-term degradation in system performance by causing undesirable wastage of valuable disk space at the peers.

If migration is used, reasonable amount of load-balancing can be guaranteed, but data availability may decrease as the peer to which data have been migrated may leave the system. Moreover, assuming data item D_1 is being accessed frequently at peer P_1, it may be migrated to another peer P_2. The implication is that every query for D_1 at P_1 will have to incur extra overhead (more response time) in accessing the data from P2. Moreover, migration necessitates the maintenance of cache-consistency. Assume some data item D_1 is migrated from peer P_1 to peer P_2. Since D_1 was a hot data item, P_1's cache may still be containing a copy of D_1. So, queries for D_1 at P_1 may be answered efficiently from P_1's cache without D_1 being actually present in P_1's disk. However, any updates to D_1 at P_2's disk will *not* be propagated to P_1's cache.

Interestingly, sensitive data (e.g., financial data, credit card numbers) are hardly ever shared in P2P systems. Users typically share non-sensitive data (e.g., mp3s, video files) and whether such files are obsolete or recent often does *not* matter to the user. Hence, the question arises: how important is it to maintain replica-consistency or cache-consistency regularly? Also, variation in available disk space among peers has implications for migration/replication.

Run-time decision-making

For both intra-cluster and inter-cluster load-balancing, we propose that the **run-time** decision concerning migration/replication should be made as follows. Every cluster leader monitors its peers' availability over a period of time. If the probability of a peer P_1 leaving the system is very low, hot data should be migrated to P_1, otherwise hot data should be replicated for availability reasons. Note that migration/replication will *only*

[3]If the same query is issued from different peers, randomness may guarantee a certain amount of load-balancing.

be done subject to disk space constraints at the destination peer. Moreover, large data items shall *only* be replicated (if necessary) at peers whose disk capacities are much larger than that of the size of the large data items.

Incidentally, available disk space may vary significantly among peers. We adopt the following strategy.

- 'Pushing' non-hot data (via migration for large-sized data and via replication for small-sized data) to large capacity peers as much as possible.
- Replicating small-sized hot data at small capacity peers (smaller search space expedites search operations).
- Large-sized hot data are migrated to large capacity peers only if such peers have low probability of leaving the system, otherwise they are replicated at large capacity peers, an upperlimit being placed on the number of replicas to save disk space.

In case of replication, each peer P_i keeps track of the set D of data items replicated at itself. Periodically, P_i checks the number of accesses N_k for the last time interval on each item in D to ascertain which items are still hot. Those items, for which N_k falls below a pre-specified threshold, are deleted since those items may not be hot anymore, thereby eliminating the need for their replication. Note that the primary copy of these replicas still remain at the peer which initiated the replication, thereby implying that the original data item is *not* deleted. Periodic deletion of replicas results in more available disk space and is important for providing system scalability over time.

In contrast, for migration, peers do *not* distinguish between migrated data D_m that they contain and their own data. Consequently, even if the number of accesses to D_m is low, D_m will *not* be deleted since it does not result in wastage of disk space. Moreover, since usually non-sensitive data are shared in P2P systems, we propose that lazy replica updates (if necessary at all) via piggybacking with other messages should be performed. Notably, even after a data item has been migrated away from a peer P_i, it may still remain in P_i's cache. We believe that no harm is practically done if the user retrieves obsolete non-sensitive data from a peer's cache. Hence, while migrating data, we do *not* specifically enforce consistency between cache and disk, thereby minimizing cache-consistency maintenance overheads.

Intra-cluster load-balancing

In case of intra-cluster load-balancing, decisions concerning when to trigger the load-balancing mechanism, hotspot detection and the amount of data to be migrated or replicated are critical to system performance. We shall now analyze two possible approaches towards such decision-making.

1. **Centralized Decision-making:** In this approach, each peer periodically sends its workload statistics to its cluster leader. Load-balancing is initiated when the cluster leader detects a load imbalance in the cluster.
2. **Distributed Decision-making:** Each peer checks the loads in its neighbouring peers to determine whether it is overloaded. In case it is overloaded, it initiates load-balancing. A variant of the distributed approach is to divide the set of peers into clusters such that each peer knows *only* about the workload statistics of the peers in its own cluster.

A major drawback of the distributed approach is that unlike the centralized approach, it does *not* take into account the workload statistics of the whole system, while initiating load-balancing. Hence, it may result in some unnecessary and unproductive migrations/replications and this is clearly undesirable. However, in the distributed approach and its variants, if a peer wishes to make the load-balancing decision based upon the current loads of *all* the other peers, it has to acquire knowledge about the current workload statistics of *all* the other peers and this is clearly undesirable since it results in significant communication overhead. Keeping these points in mind, we adopt a centralized approach towards intra-cluster decision-making.

Intra-cluster load-balancing has been well researched in the traditional domain [2, 7, 12], but for P2P systems, we should also take into account varying available disk capacities of peers. Our strategy is as follows. The cluster leader (say C_i) periodically receives information concerning loads L_i and available disk space D_i of the peers and initially creates a list *List* by sorting the peers based *only* on L_i such that the first element of *List* is the most heavily loaded peer. Assume there are n elements in *List*. Among the last $\lceil n/2 \rceil$ peers in *List*, the peers whose respective values of D_i are less than a pre-specified threshold are deleted from *List*. Then load-balancing is performed by migrating or replicating hot data from the first peer in *List* to the last peer, the second peer to the second-last peer and so on. Data are only moved (migrated or replicated) if the load difference between the peers under consideration exceed a pre-specified threshold.

Observe that C_i checks for load imbalance only at periodic time intervals and **not** whenever any peer joins or leaves the system. Any load imbalance caused by some peers joining/leaving the system will be corrected by C_i only at the next periodic time interval. Since peers may join/leave the system frequently, we believe that performing load-balancing every time a peer joins/leaves will result in undesirable thrashing conditions. (Thrashing implies that peers spend more time on load-balancing than for doing useful work.)

Inter-cluster load-balancing

To prevent load imbalance among clusters, inter-cluster load-balancing becomes a necessity. We propose that

such load-balancing should be performed *only* between neighbouring clusters by means of collaboration between the cluster leaders, the reason being that moving data to distant clusters may incur too high a communication overhead to justify the movement.

Cluster leaders periodically exchange load information *only* with their neighbouring cluster leaders. If a cluster leader α detects that its load exceeds the average loads of the set β of its neighbouring cluster leaders by more than 10% of the average load, it first ascertains the hot data items that should be moved and sends a message concerning each hot data item's space requirement to each cluster leader in β in order to offload some part of its load to them. The leaders in β check the available disk space in each of their cluster members and if their disk space constraint is satisfied, they send a message to α informing it about their total loads and their total available disk space. α sorts the willing leaders of β in $List_1$ such that the first element of $List_1$ is the least loaded leader.

Assume the hot data items are numbered as h_1, h_2, h_3, h_4... (h_1 is the hottest element). Let the number of willing peers in β and the number of hot data items be denoted by b and h respectively. If $b < h$, h_1 is assigned to the first element in $List_1$, h_2 is assigned to the second element and so on in a round-robin fashion till all the hot items have been assigned. If $b \geq h$, the assignment of hot data to elements of $List_1$ is done similarly, but in this case some elements of $List_1$ will *not* receive any hot data. We shall subsequently refer to this technique as **L_assign**.

After the hot data arrives at the destination cluster's leader, the leader creates a sorted list $List_2$ (in ascending order according to load) of its peers and assigns the hot data to elements of $List_2$ using the **L_assign** algorithm.

5. PERFORMANCE STUDY

This section reports the performance evaluation of our proposed techniques. Note that we consider performance issues associated *only* with inter-cluster load-balancing since a significant body of research work pertaining to efficient intra-cluster load-balancing algorithms already exists. We specifically study the performance of our proposed scheme with variations in the workload skew. For the sake of convenience, we shall henceforth refer to our proposed scheme of performing load-balancing via migration/replication of data as **LBMR** (load-balancing via migration/replication) and the policy of *not* performing any load-balancing as **NLB** (no load-balancing).

5.1 Experimental setup

Our test environment comprises a set of PCs, each of which is a 800 MHz Pentium-III processor running the Solaris 8 operating system. Each PC has 128 MB of main memory and total disk space of 18 GB. We have used 4 PCs for our performance study.

Each cluster is modeled by a PC in our experiments and the decision-making associated with a given cluster is simulated as being performed by the cluster leader of the given cluster. A moment's thought indicates that for our experiments, each cluster leader is representative of its entire cluster. The implication is that there are 4 neighbouring clusters among which we attempt to provide inter-cluster load-balancing. Note that this is in accordance with our objective of focussing mainly on inter-cluster load-balancing. Additionally, we simulated a transfer rate of 1 MB/second among the respective clusters.

Owing to space constraint arising from other users using the system, we were able to use only 10 GB[4] of disk space in each PC for the purpose of our experiments. Moreover, owing to such constraints associated with available disk space, the decision-making step in our algorithm concerning the possible options of migration and replication always chose migration. Hence, our experimental results primarily reflect the improvements in system performance owing to migration *only*. However, in the near future, we also intend to study the impact of replication since we expect that replication can facilitate further improvement in system performance.

For our experiments, we have used real mp3s with the objective of ensuring that our experiments are in consonance with real-life scenarios as far as possible. We used the Kazaa P2P application software for downloading the real mp3s.[5] The Kazaa application software can be downloaded from http://www.kazaa.com. The sizes of the respective mp3s used in our experiments ranged from 1.8 MB to 3.5 MB. In all of our experiments, the system checks the load situation periodically.

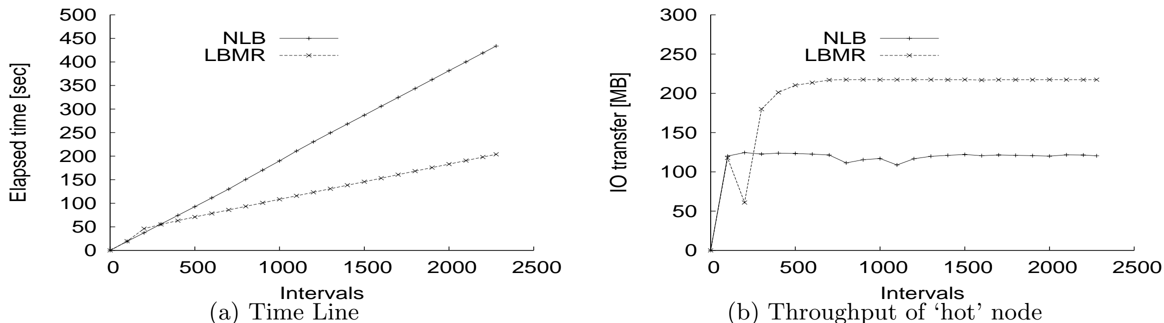
In order to model skewed workloads, we have used the well-known Zipf distribution to decide the number of queries to be directed to each cluster. Note that this is only an approximate manner of generating skewed workloads since the actual load imposed on a peer depends not only upon the number of queries directed to the peer, but also on the individual *sizes* of the respective queries. We modified the value of the *zipf factor* to obtain variations in workload skew. A value of 0.1 for the zipf factor implies a heavily skewed workload, while a value of 0.9 indicates extremely low skew in the workload.

5.2 Performance of our proposed scheme

Now let us investigate the effectiveness of our proposed scheme in improving the system performance. For this purpose, an experiment was performed using 50000 queries, each query being a request for one of the real mp3 files that we had earlier downloaded. A Zipf distribution was used over 4 buckets to decide the number of queries that were to be directed to each of the 4 clusters, the

[4]Each PC stored approximately 10 GB of mp3 files.

[5]To the best of our knowledge, *none* of the real mp3s, which we had downloaded, violates any existing copyright laws.



(a) Time Line (b) Throughput of 'hot' node

Figure 3: Performance of our proposed scheme

value of the zipf factor being set at 0.1, which indicates a highly skewed workload. Figures 3 and 4 depict the experimental results.

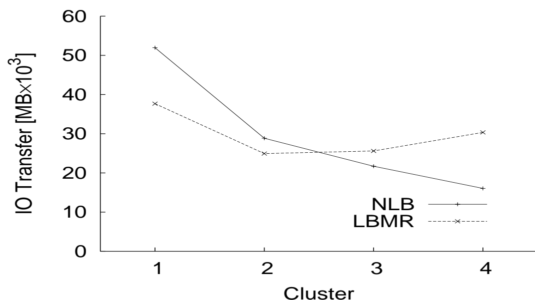


Figure 4: Load-balancing

Figure 3a displays a time line indicating the progress of the queries over time by presenting the wall-clock execution times of queries as a function of the time intervals during which the queries were executed. Figure 3b presents the throughput of the hot node as a function of the time intervals pertaining to the query executions. From Figure 3, we observe that initially during certain intervals, the performance of LBMR is worse than that of NLB. This slight degradation in performance occurs owing to migration-related disturbances. However, once the migration has been completed, LBMR significantly outperforms NLB. This is possible because of the improvement in the throughput of the hot node as demonstrated by Figure 3b. Since for parallel systems, response time of user queries is dictated by the hot node, such improvements in throughput of the the hot node is extremely desirable. Notably, such improvement in throughput is made possible due to the reduction in the load of the hot node as a result of the effective load-balancing provided by LBMR.

Figure 4 manifests the load-balancing capabilities of LBMR by indicating the respective loads at the 4 clusters during the entire time interval when the experiment was conducted. Just to recapitulate, we have used the number of Megabytes retrieved as a measure of load. Figure 4 indicates that LBMR is capable of distributing the load more evenly than NLB (especially, reducing the loads of the hot cluster, namely, cluster 1). In summary, LBMR is effective in correcting the degradation in system performance owing to the overloading of certain clusters by a skewed query distribution.

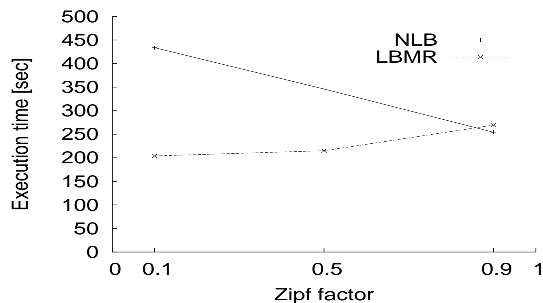
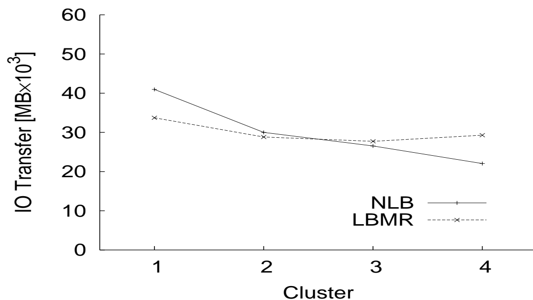


Figure 5: Variation in workload skew

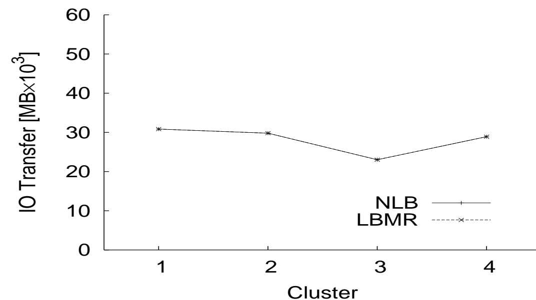
5.3 Variations in Workload Skew

Now we shall examine the performance of LBMR for varying skews in the workload. For this purpose, we distributed a set of 50000 queries (each query is a request for an mp3 file) into 4 buckets (corresponding to 4 clusters) using zipf factors of 0.5 and 0.9 to model medium-skewed workload and low-skewed workload respectively.

Figure 5 depicts the wall-clock completion time of *all* the queries when the zipf factor was varied. Interestingly, the gain in execution time is significantly more in the case of highly skewed workloads. This gain keeps diminishing as the query skew diminishes, till at some point, there is *no* significant gain at all. This occurs because as the workload skew decreases, the need for load-balancing also reduces owing to the query pattern itself contributing to load-balancing. Note that at a value of 0.9 for the zipf factor, there is no significant difference in performance between LBMR and NLB since the workload in this case was too lowly skewed to necessitate migrations. Incidentally, LBMR performs slightly worse than NLB for lowly skewed workloads primarily owing to overheads incurred in making the decision that



(a) Zipf factor=0.5



(b) Zipf factor=0.9

Figure 6: Load-balancing for medium and lowly skewed workloads

the skew is too low to necessitate migrations. However, we believe this is a small price to pay as compared to the big gain achieved by LBMR in the case of highly skewed workloads.

Figure 6 manifests the load-balancing capabilities of LBMR in case of medium and lowly skewed workloads. Figure 6a demonstrates that LBMR performs reasonably well for medium-skewed workloads, especially in reducing the load of the hot cluster. From Figure 6b, we observe that the performance of LBMR and NLB is comparable since no migrations were performed.

The performance study indicates that LBMR provides significant improvement in system performance for heavily skewed workloads. In case of medium-skewed workloads, LBMR remains effective, while for lowly skewed workloads, the load-balancing performed by LBMR is *not* worse than that of NLB. In essence, LBMR is sensitive to changing workload distributions and adapts very well indeed.

6. CONCLUSION

The sheer size of P2P systems and the dynamic environments in which they are deployed makes efficient data management in such systems a challenging problem. Dynamically changing popularities of data items and skewed user query patterns necessitate a load-balancing mechanism to facilitate reduced response times for user queries. In order to make huge P2P systems manageable, we have viewed a P2P system as comprising clusters of peers and addressed both intra-cluster and inter-cluster load-balancing via migration and replication. Moreover, we have also proposed a technique for automatic clustering of peers. To our knowledge, this work is one of the earliest attempts at addressing load-balancing via both *online* data migration and replication in P2P environments.

To this end, we believe that our contributions have addressed some of the relevant issues associated with load-balancing in P2P systems. In the near future, we wish to extend this work by performing a detailed performance evaluation with the objective of identifying possible avenues for improving our proposed LBMR scheme.

7. REFERENCES

[1] Y. Breitbart, R. Vingralek, and G. Weikum. Load control in scalable distributed file structures. *Distributed and Parallel Databases*, 4(4):319–354, 1996.

[2] G. Copeland, W. Alexander, E. Boughter, and T. Keller. Data placement in Bubba. *SIGMOD Record ACM*, 17(3):99–108, 1988.

[3] A. Crespo and H. G. Molina. Routing indices for Peer-to-peer systems. *Proc. ICDCS*, 2002.

[4] D. H. J Epema, M. Livny, R. V. Dantzig, X. Evers, and J. Pruyne. A worldwide flock of Condors : Load sharing among workstation clusters. *Journal on Future Generations of Computer Systems*, 12, 1996.

[5] Gnutella. http://www.gnutella.com/.

[6] Kazaa. http://www.kazaa.com/.

[7] A. Mondal, M. Kitsuregawa, B.C. Ooi, and K.L. Tan. R-tree-based data migration and self-tuning strategies in shared-nothing spatial databases. *Proc. ACM GIS*, 2001.

[8] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. *Proc. IFIP/ACM*, 2001.

[9] T. Stading, P. Maniatis, and M. Baker. Peer-to peer caching schemes to address flash crowds. *Proc. IPTPS*, 2002.

[10] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *Proc. ACM SIGCOMM*, 2001.

[11] R. Vingralek, Y. Breitbart, and G. Weikum. SNOWBALL: Scalable storage on networks of workstations with balanced load. *Distributed and Parallel Databases*, pages 117–156, 1998.

[12] Gerhard Weikum, Peter Zabback, and Peter Scheuermann. Dynamic file allocation in disk arrays. *In Proc. ACM SIGMOD*, pages 406–415, 1991.