CONCURRENT IMAGE QUERY USING LOCAL RANDOM WALK WITH RESTART ON LARGE SCALE GRAPHS

Yinglong Xia[1]*, Jui-Hsin Lai*[1]*, Lifeng Nai*[2]*, and Ching-Yung Lin*[1]

[1]IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598, USA
[2]Georgia Institute of Technology, Atlanta, GA 30332, USA
{yxia,larrylai,chingyung}@us.ibm.com, lnai3@gatech.edu

ABSTRACT

Efficient image query is a fundamental challenge in many large scale multimedia applications, especially when handling many queries concurrently. In this paper, we proposed a novel approach called graph local random walk for high performance concurrent image query. Specifically, we organize the massive images set into a large scale graph using graph database, according to the similarity between images. A heuristic method is utilized to map each query image to some vertex in the graph, followed by a local search to refine the query results using an alternative of local random walk on graph. The local random walk process is essentially a weighted partial traversal in the local subgraphs for finding a better match of the query images. We organize the graph of the image set in a parallelization amenable approach, so that a set of partial graph traversal for local random walk can be performed concurrently, taking the advantage of the multithreading capability of processors. We implemented the proposed method in state-of-the-art multicore platforms. The experimental result shows that the graph local random walk based approach outperforms baseline methods in terms of both throughput and scalability.

Index Terms— Graph, image query, parallel, multicore

1. INTRODUCTION

Multimedia increasingly creates Big Data. Similar to many big data applications, big multimedia problems can involve a massive collection of interconnected entities, which is naturally represented as graphs. Therefore, it is reasonable to leverage the advances in high performance graph computing and graph databases for solving big multimedia problems efficiently. We take the concurrent image query problem as an example to illustrate the role of graph computing in multimedia. The proposed techniques in this paper can be easily extended to various other big multimedia applications.

`Problem definition`: Given a set of query images, for each query image, we concurrently find the image with the highest similarity from a collection of images, a.k.a the *image reservoir*, in near real-time, where the image reservoir can be dynamically updated. By *concurrent*, we mean that all the queries should be handled in parallel to enhance the throughput. Figure 1 illustrates this process.

`Challenges`: (1) *High data volume* makes it impossible to perform concurrent image query in the native way, i.e., performing an exhaust search by comparing each query image against every image in the reservoir. Some hierarchical method or divide-and-conquer solution should be considered. (2) *Concurrency* requires us to explore parallel computing elements of platforms in the solutions, making it scale up and/or out. (3) *Irregularity* in the entity interconnection makes it challenging to index the images while preserve the similarity in index locally.

`Contributions` in this paper include: (1) *High data volume* is addressed by a heuristic that maps a query image to a subgraph, followed by a local random walk on the subgraph to refine the searching result; (2) *Concurrency* is handled by a highly efficient scheduler design that performs local search on graph in parallel; (3) *Irregularity* is resolved by the graph database based solution that organizes the images from a reservoir into a large scale graph based on their similarities. We implemented the proposed technique on parallel computing platforms and achieve superior performance compared to baseline methods.

The paper is organized as follows: We discuss the background in Section 2 and then present the image reservoir using a graph database in Section 3. The local random walk with restart on graph is analyzed in Section 4, followed by the cache-aware scheduling in Section 5. Experimental study is discussed in Section 6. Section 7 concludes this paper.

2. BACKGROUND AND RELATED WORK

Let's denote a collection of images by $V = \{v_1, v_2, \cdots, v_n\}$, where v_i represents an image. Define a metric s_{v_i,v_j} as any similarity measurement between two images. s_{v_i,v_j} can be implemented in several ways, up on the specific application. See Section 3 for details. Given a threshold ϵ, an *image reservoir* based on image set V can be defined as a graph $\mathcal{G}(V, E)$, where $E = \{(v_i, v_j) | s_{v_i,v_j} > \epsilon, v_i, v_j \in V\}$. Given a set of

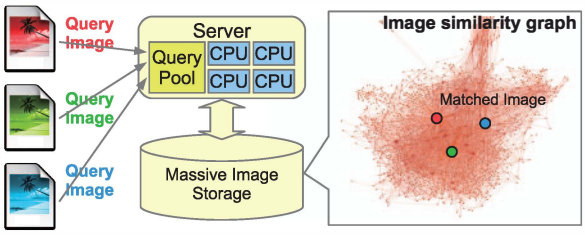

Fig. 1. System View of Image query

query images $\tilde{V} = \{v_x\}$, the *concurrent image query* problem is to find a vertex:

$$v_y = \arg\max_{v_j \in V} s_{v_x, v_j}, \ \forall v_x \in \tilde{V} \quad (1)$$

Note that v_x may or may not belong to V. In case we have $v_x \notin V$, then the query is approximate. Given a starting vertex $v'_y \in V$ reasonably similar to v_x, then v_y can be possibly identified using a *local search* on the graph [1].

`Random walk` has been studied for local search [2][3]. Basically, given a vertex $v_i \in V$ as the starting point, a particle can move from v_i along the edges in $\mathcal{G}(V, E)$. If the particle is at vertex $v_i \in V$ at some time, then it will be at a neighbor of v_i, say v_j, at the next time; the neighbor is chosen randomly, in proportion to the transition probability p_{v_i, v_j} associated to $(v_i, v_j) \in E$. The expectation time for the particle to move from v_i to $v_{k \neq i} \in V$ is called the *hitting time* $t_{v_i \to v_k}$. Hitting time has been well studied for *finite graphs*.

`Random walk on infinite graphs` imposes fundamental challenges [4]. Due to the massiveness of the graph considered in our scenario, it is not efficient to perform the random walk over the entire graph $\mathcal{G}(V, E)$. Instead, it is performed within a local subgraph $\mathcal{G}_{v_i}(V', E')$ starting from v_i. The local subgraph is *unbounded*, since we can not predetermine the scale of $\mathcal{G}_{v_i}(V', E')$. If we have $|V'| \ll |V|$ which is typically true, this is approximately equivalent to the random walk on infinite graphs. It is worth noting that the hitting time $t_{v_i \to v_{k \neq i}}$ between vertices is generally infinite. For example on an infinite line, the expected time to get from 0 to 1 is infinite.

`Random walk with restart` (RWR) addresses the challenge for random walk on infinite graphs. If we allow the random walk to restart at v_i, then we might expect the hitting time with restarts to be finite, since in this way it eliminates the possibility a walk will wander to far off towards infinity [5]. The RWR algorithm has been discussed in several literature [1][4], although we have not found any multimedia application of RWR with emphasis on system throughput. We propose concurrent RWR based solution as a promising technique for large scale image query in graph database.

`Concurrent search` is more than conducting multiple local searches independently. For example, assume two local searches are performed on $\mathcal{G}(V, E)$ starting at v_x and v_y, respectively, where each visits a set of vertices that span a subgraph, denoted by $\mathcal{G}_x(V_x, E_x)$ and $\mathcal{G}_y(V_y, E_y)$. If $V_x \cap V_y \neq \emptyset$, then scheduling the two searches onto processors (cores) with shared cache leads to improved performance, due to the data locality. Similarly, when a processor was allocated a set of local search tasks on $\mathcal{G}$, it is more efficient to perform those with overlapped search areas successively [6]. There is some existing work discussing parallel/distributed random walk [4][7]; however, to the best of our knowledge, none of them can be directly utilized for image query based on random walk with restart on infinite graphs. Therefore, we develop a cache-aware scheduler for concurrent image query. We particularly focus on scaling up image query on multi-core/manycore processors, although our techniques can be extended to distributed computing environment as well.

3. MASSIVE IMAGES AS A GRAPH

Recent advances in graph database provide novel approaches for organizing multimedia dataset. For image search applications, the geometry of photo capture among pictures can be modeled in the graph connection to increase searching speed and accuracy [8]; Visual features (e.g., BoW) or text to find the approximate nearest images through the effective feature index (e.g., LSH) can also be modeled in a graph structure for increasing accuracy [9].

In our scenario, a large scale graph is constructed to represent a massive set of human face images, where each vertex represents an image while each edge indicates the interconnection between two images. The interconnection can be defined in various ways. In our case, an edge means that the similarity of the two images is above a pre-defined threshold ϵ. The detailed discussion for similarity measure among face images is beyond the scope of this paper. For the sake of completeness, we briefly present the method that we used. Our method combines the grid based scale invariant feature transform (Grid-SIFT) [10] and geometric feature matching.

`Keypoint extraction` is performed with respect to a human face image in 4 steps: 1) We detect face region and normalize it to a fixed size; 2) We divide the face region into four subregions, i.e., forehead, eye, nose, and mouth (see Figure 2(a)). 3) We use the Harris corner detection to identify keypoints and then 4) apply SIFT descriptor for keypoint descriptions. Each keypoint is represented by a vector of 128 elements.

`Keypoint matching` gives the similarity between two images, which is defined as the sum of the similarity of all the corresponding subregions between the two images. Within a subregion r, we have a set of keypoints extracted from the raw images v_x and v_y. We define matrix $A_r = \{a_{ij}\}$ where a_{ij} is the negative exponential function of the sum of square difference (SSD) between the i-th keypoint and the j-th keypoint from the corresponding subregions in v_x and v_y, representing the similarity between the two keypoints. The

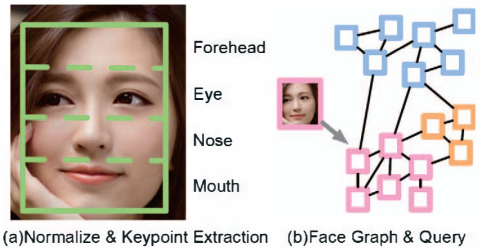

Fig. 2. (a) Normalize a face region and partition it into four subregions (grids); (b) Image reservoir built by image similarities. The vertex color shows partitions.

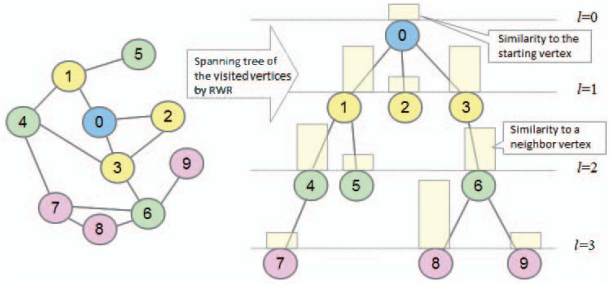

Fig. 3. Local Random Walk on Graph

keypoint matching is to find a *permutation matrix* X_r for exchanging the columns (or rows) in A_r, so that the i-th keypoint in v_x corresponds to the i-th keypoint in v_y, and $\sum_i a_{ii}$ is maximized. Therefore, we calculate the permutation matrix X_r by a linear programming:

$$\max_X \ \mathrm{tr}(A_r X_r)$$
$$s.t.\ X_r\vec{1} = \vec{1},\ X_r^T\vec{1} = \vec{1},\ X_r \geq 0 \qquad (2)$$

where $\vec{1}$ is a vector of all 1s and the constraints ensures X is a permutation matrix [11]. The *geometric feature matching* can also be applied to X_r for preserving matching pairs geometry relationship, to avoid matching the left eye in v_x to the right eye in v_y. Denoting R the number of subregions in an image, the similarity of the two images can be obtained by:

$$s_{v_x,v_y} = \sum_{r=1}^{R} s^r_{v_x,v_y} = \sum_{r=1}^{R} A_r X_r. \qquad (3)$$

`Image Reservoir` $\mathcal{G}(V,E)$ is built according to the image similarities. As shown in Figure 2(b), each vertex represents a human face image and each edge links two similar images. The image reservoir can be partitioned, possibly corresponding to the clustering of images. We illustrate the partitions of the graph using different colors in Figure 2(b). Various metrics can be utilized for the partitioning. In our scenario, it is the difference of the first and second largest dominant directions of the SIFT keypoints of a face image.

`Image query` was defined in Section 2. We extract the keypoints from a query image in the same approach as we process the images in the reservoir. Then, it is straightforward to categorize the query image into a partition in $\mathcal{G}(V,E)$ using the dominant directions of the SIFT keypoints. A local search (see Section 4) can be conducted starting from some representative vertices in this partition, such as the central vertex of a partition, to refine the result.

4. GRAPH LOCAL RANDOM WALK WITH RESTART

The local search is implemented by a random walk with restart (RWR) on a graph, where the graph topology comes from the image reservoir $\mathcal{G}(V,E)$. Given a query image v_x, a starting vertex v_i, the RWR will traverse a subgraph $\mathcal{G}_{v_i}(V',E')$. Denote a vector $\mathbf{q}^t = (q_1, q_2, \cdots, q_{|V'|})$, where t is a time stamp and q_i is the probability that a particle remains in vertex $v_i \in V'$. Initially, since all walkers start from v_i, we have $\mathbf{q}^0 = (0, \cdots, 0, 1, 0, \cdots)$, i.e., only $q_i = 1$. These particles may remain in v_i at probability $p_{i,i} = \frac{1}{Z} s_{v_i,v_i}$, or walk to its neighbor, say v_j, at probability $p_{i,j} = \frac{1}{Z} s_{v_i,v_j}$, where $Z = s_{v_i,v_i} + \sum_{\forall(i,k)\in E'} s_{v_i,v_k}$ is a normalizer. Let $P = \{p_{i,j}\}$ denote the transition matrix, we have $\mathbf{q}^{t+1} = P^T\mathbf{q}^t,\ t = 0, 1, \cdots$. After the first iteration, the particles originally at v_i possibly walk away, so we may have multiple elements in $\mathbf{q}^1$ be nonzero. As time goes by, $\mathbf{q}^t$ tells us the probability that the particles stay in each vertex. Those with high probabilities should be returned as the local search result. In simplicity, we return the vertex corresponding to the maximum element in $\mathbf{q}^t$ as the search result. In RWR, we dynamically compute the transition probabilities P on the edges according to the similarity of the query image and the images in the reservoir, so that the RWR can find the most likely matched image to v_x.

Due to the agnostic to the subgraph $\mathcal{G}_{v_i}(V',E')$ in local search, we cannot obtain $\mathbf{q}^t|_{t\to\infty}$ by the eigen decomposition of P (i.e., solving $\mathbf{q} = P^T\mathbf{q}$), as what we did for many random walks on finite graphs. Instead, we simulate the walk of the particles. Note that the particle walk on $\mathcal{G}_{v_i}$ is essentially a partial traversal of the subgraph. By partial traversal, we mean that some vertices may not be visited due to the low transition probabilities. Thus, starting from v_i, we can construct a tree shown in Figure 3 to illustrate the partial traversal level by level. Each vertex is associated with a similarity to the starting vertex s_{v_k,v_x} shown as a bar in the figure.

We extend the concept of restart in RWR to allow a random walk restart from *any* visited vertex, instead of the original starting vertex. When the random walk reaches a new (unvisited) vertex, we expand the tree by adding the vertex as a leave; otherwise, we restart from the visited vertex. For example, if a particle walks from vertex 4 to vertex 3 in the subgraph, it is equivalent that we restart a new random walk starting from vertex 3.

There is a trade-off between the accuracy and efficiency

in the RWR based local search. Specifically, the larger the local search scope (i.e. the size of subgraph $\mathcal{G}_{v_i}(V', E')$) is, it is more likely to find the best matching; but it consumes higher execution time. Given a vertex v_z in the image reservoir $\mathcal{G}(V, E)$, we estimate the execution time using the *hitting time* from a starting vertex v_i to any vertex v_z, i.e., $t_{v_i \to v_z}$. For the simplicity, we assume $\mathcal{G}_{v_i}(V', E')$ is an infinite graph, where each vertex has equal degree d. Such a tree is known as d-regular *Cayley tree*, denoted by T^d. Note that T^d is symmetric, so we have $t_{v_i \to v_z} = t_{v_z \to v_i}$. Assume v_z is at level k in T^d. Let T^d_k denote the truncated tree that contains the first k levels of T^d with $d-1$ loops for each vertex at level k. We can see that the hitting time with restarts from level k to the starting vertex is the hitting time on T^d_k [5]. Going from level $j+1$ to an adjacent vertex u at level j requires itself a *whirling tour* on subgraph T^d_{k-j} rooted at u [5]. Note T^d_{k-j} has $\sum_{i=0}^{k-j-1}(d-1)^i$ edges and $(d-1)^{k-j}$ loops, and each edge is used twice and each loop once. Let W_j be the length of a whirling tour on T^d_k from level $j+1$ to an adjacent vertex u on level j. According to [5], we have:

$$W_j = 2 \cdot \left(\sum_{i=0}^{k-j-1} (d-1)^i \right) - 1 + (d-1)^{k-j} \qquad (4)$$

Recall that we allow a random walk restart at any visited vertex, which can be viewed as restarting at the root and then walking to the vertex. Therefore, we estimate the upper bound of hitting time $t_{v_i \to v_z}$ by summing W_j over all levels:

$$\begin{aligned} t_{v_i \to v_z} &\leq \sum_{i=0}^{k-1} W_i \\ &= \frac{(d-1)^k + 1 - (d-1)(d + (d-1)k - 1)}{(d-2)^2} \end{aligned} \qquad (5)$$

Now, given the acceptable searching time, we can estimate how faraway our RWR should explore in $\mathcal{G}(V, E)$.

5. CACHE-AWARE SCHEDULING OF CONCURRENT QUERIES

Computational characteristics of the proposed solution for concurrent image must be studied for providing an efficient implementation. The two major primitives involved in the proposed solution are the *graph random walk with restart* (RWR) and the *image similarity estimate* (see Sections 3 and 4, respectively). Although the two are closely related, they show significantly different computational characteristics. Therefore, we separate the two workload as illustrated by Figure 4, both are processed concurrently. Note that within the same query, the similarity must be estimated before performing RWR.

- RWR is a typical *graph computing*, where the data access patterns are highly irregular, which normally results in poor cache performance. We parallelize the graph computing utilizing a graph database called IBM System G, which implements native support for various graph operations [12]. Besides, we explore lock-free data structures and a cache-aware scheduler to address the challenge.

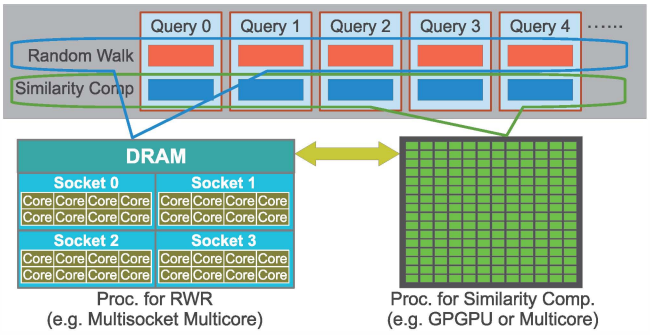

Fig. 4. Graph random walk and similarity estimate are processed in separate parallel units, both in concurrent manner.

- Similarity estimate is a traditional *scientific computing*, involving matrix/vector calculations (see Eq. 2), which typically shows regular memory access patterns and satisfactory cache performance. Since the matrices and vectors are straightforward to be evenly partitioned, parallelization of such workload is trivial and well studied [13]. We dedicate a set of cores in a server to parallelize the similarity estimate. Some advanced processors, such as GPGPUs and/or Intel Phi, can be exploited to further speed up the execution.

`Lock-free concurrent RWR` data structure is utilized for processing RWRs in parallel. Since the traversed subgraphs of the two RWRs, say $\mathcal{G}_{v_i}(V_i, E_i)$ and $\mathcal{G}_{v_j}(V_j, E_j)$, may overlap, updating the probability $\mathbf{q}^t$ (see Section 4) for overlapped vertices $v \in V_i \cap V_j$ concurrently can cause ambiguous results a.k.a. *data race*. Although this can be eliminated using mutex locks, it leads to increasing synchronization overhead as the number of cores increases. Our proposed lock-free data structure spawns up to P separate slots, i.e. $\mathbf{q_1}^t, \cdots, \mathbf{q_P}^t$, for the overlapped vertices, where P is the number of threads. $\mathbf{q_i}^t$ is padded to avoid $\mathbf{q_j}^t$, $j \neq i$ being in the same cache line that can cause *false sharing*. Since there are only P concurrent threads running in the system, we have at most P concurrent RWRs, each now having its own $\mathbf{q}^t$ to work on. Therefore, data race is completely avoided without using any lock.

`Cache-aware scheduling` allocates a set of image queries to a set of processors (cores) with shared and/or separated caches. Cache-aware scheduling aims at maximizing the re-use of the data that have been loaded into the cache. As pointed out earlier this section, two RWRs $\mathcal{G}_{v_i}(V_i, E_i)$ and $\mathcal{G}_{v_j}(V_j, E_j)$ may overlap, i.e. $V_i \cap V_j \neq \emptyset$. Scheduling the two RWRs onto the same processor successively, or onto two

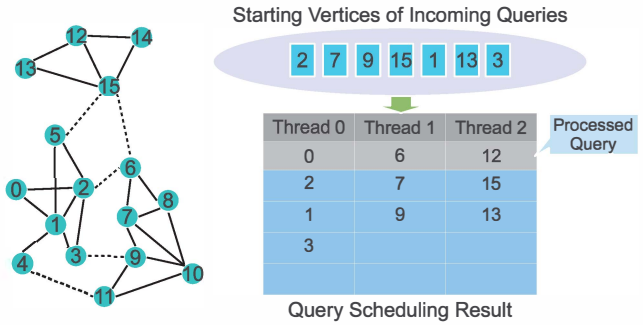

Fig. 5. Query scheduling example

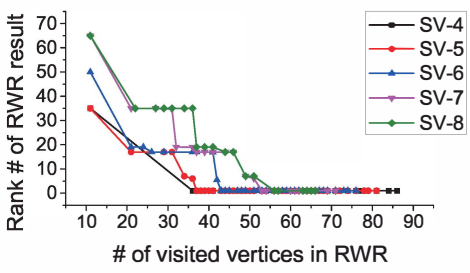

Fig. 6. Accuracy evaluation of image query based on graph local random walk

processors with shared cache in parallel, help re-use the data in cache. The former is called *intra-thread locality* and the latter called *inter-thread locality* [6]. For estimating the locality of queries, we let a RWR mark the vertices it walks. The mark on a vertex v consists of a time stamp plus the CPU socket ID, i.e. (t_v, s_v), where we assume all the cores within a socket have the shared cache (e.g., L2 or L3 data cache). (1) To improve the inter-thread locality, when a new query comes, we first determine its starting vertex (Section 3), say u, and then check if u is marked. If not, the query is assigned to any processor that is not overloaded; otherwise, it is assigned to any processor in socket s_u, since the neighborhood of u was last loaded into the cache of s_u. (2) To improve the intra-thread locality, each processor handles the assigned queries in reverse order of the time stamp of their starting vertices, since the most recent data is more likely cached. In addition, to avoid any processor being overloaded, we perform *work stealing* periodically to balance the workload [14].

6. EXPERIMENTS

We implemented the proposed solution using C++ with PThreads on top of the IBM System G Native Store, a high performance graph database [12]. The code was compiled using g++ 4.4.6 with -O3 optimization in CentOS 6.3 on an IBM BladeCenter multi-socket multicore server. The server has 4 Intel Xeon E7-4830 processors running at 2.13 GHz and the memory size is 256 GB. Each processor contains 8 cores with hyperthreading enabled. So, 64 concurrent hardware threads are supported.

`Experiment Datasets` in our evaluations consists of: (1) a *synthetic dataset* that was converted into a graph with 5000 vertices and 100,000 edges. This dataset is used for verifying the graph local random walk with restart (RWR) in terms of correctness and effectiveness, so we precomputed the image similarities; (2) a *real-world image reservoir* provided by the ISVision Co. for the ICME'2014 Industry Grand Challenge [15], including 5978 photos and 24 videos from 336 persons. The corresponding graph has 5978 vertices, 89,206 edges, and 45 partitions. An additional set of 1024 testing images was taken from those persons for generating queries. This dataset was utilized to verify the efficiency and scalability of the concurrent graph queries.

`Accuracy evaluation` results are shown in Figure 6, where we evaluated the proposed RWR technique on the synthetic dataset with various parameters, such as the *starting vertex* (SV) for a query. To estimate the impact of the SV, for each query image, we intentionally chose multiple SV positions with different distance to the target vertex, i.e., the best matched image. Figure 6 shows the experimental results where the SVs were 4 to 8 hops away from the target vertex, denoted by `SV-4` to `SV-8`. To measure the accuracy of the RWR, we introduced the metric called **rank number**, which was calculated as follows: We first sorted all the vertices from the graph into a list by their similarities to the query image in descending order. Then, we performed the query and identified the rank of the returned image in the above list.

Figure 6 also illustrates the relationship between the number of visited vertices and accuracy. It shows that the more vertices get visited during the RWR, the better accuracy was achieved. Moreover, if the SV is closer to the target vertex, less vertex visits occurred. Such observations are consistent with our intuition. For example, 36 vertices were visited to reach the target vertex for SV-4, but 51 vertices were visited to reach SV-8. The experimental results show that RWR achieved the best matched result with only a few dozens of vertex visiting. That is, no more than a few dozens of similarity estimates were performed, compared to about 5000 times similarity estimates in the native baseline method, where a brute-force exhaust search was performed for finding the best matched image. Thus, our proposed RWR technique significantly reduced the similarity estimate workload by 2 orders of magnitude.

`Performance evaluation` result is shown in Figure 7. In this experiment, we established the image graph using the real-world dataset and generated 1024 queries using the test images. Two experiments were performed: (1) For the baseline method, we processed the queries using a naive single thread brute-force by comparing the query image against every image in the dataset. (2) We utilized our proposed method to execute the queries using various numbers

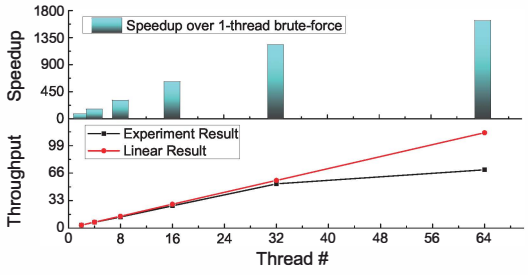

Fig. 7. Performance of the proposed concurrent image query on real-world image reservoir

of threads. The speedup over the baseline method is shown in the upper part of Figure 7 and the scalability is shown in the lower part. According to the results, our proposed technique achieved over 3 orders of magnitude in terms of speedup compared to the baseline. Even for using only 2 threads, we observed that the average time for processing a query was around 0.558 seconds; while the baseline method took 23.268 seconds. So, the speedup was 83.3×. This is because our method did not visit all the images for finding the best match. The lower part of Figure 7 shows the observed throughput compared with the ideal case (i.e., the linear speedup). The *throughput* is defined as number of queries processed per second. We can see that our proposed technique showed near linear scalability in many cases. Note that our platform has 32 CPU cores in total. Thus, when 64 concurrent threads were used, the shared hardware between hyper threads degrades the performance. In summary, the experiment results demonstrated both impressive performance speedup and scalability for our technique.

7. CONCLUSIONS

We presented a high performance framework for concurrent image query using graph database, where a massive set of images are organized into a graph. The graph links images together based on their similarities. The query images were assigned to some vertices in the graph, based on the keypoints extracted from the images. Those vertices were relatively assemble to the query images. A random walk with restart was utilized for performing a local search on the graph for refining the query results. Our solution for the concurrent image query is parallelization amenable. We designed an efficient scheduler for allocating the queries by maximizing the data locality in processor caches. The proposed method illustrates superior performance compared to the baseline method.

In future, we plan to study the SIMD for parallelization the similarity computation. Since this application involves both (partial) graph traversals and similarity computation, two tasks of different computational characteristics, we would like to port the system on heterogeneous computing platforms.

8. REFERENCES

[1] F. Yasuhiro, N. Makoto, O. Makoto, and K. Masaru, "Fast and exact top-k search for random walk with restart," *Proc. VLDB Endow.*, pp. 442–453, 2012.

[2] W. H. Hsu, L. S. Kennedy, and S.-F. Chang, "Video search reranking through random walk over document-level context graph," in *Proc. MM 2007*, pp. 971–980.

[3] S. D. Servetto and G. Barrenechea, "Constrained random walks on random graphs: Routing algorithms for large scale wireless sensor networks," in *Proc. of WSNA 2002*, pp. 12–21.

[4] M.-F. Chiang, T.-W. Wang, and W.-C. Peng, "Parallelizing random walk with restart for large-scale query recommendation," in *Proc. PODS 2010*, pp. 8:1–8:6.

[5] N. McNew, "Random walks with restarts 3 examples," 2013.

[6] L. Nai, Y. Xia, C.-Y. Lin, B. Hong, and H. Lee, "Cache-conscious graph collaborative filtering on multisocket multicore systems," in *Proc. Computing Frontiers 2014*.

[7] D. S. Atish, N. Danupon, and P. Gopal, "Fast distributed random walks," in *Proc. PODC 2009*, pp. 161–170.

[8] J. Philbin, J. Sivic, and A. Zisserman, "Geometric latent dirichlet allocation on a matching graph for large scale image datasets," *Int'l J. Computer Vision 2010*, pp. 138–153.

[9] S. Zhang, Q. Huang, G. Hua, S. Jiang, W. Gao, and Q. Tian, "Building contextual visual vocabulary for large-scale image applications," in *Proc. MM 2010*, pp. 501–510.

[10] J. Križaj, V. Štruc, and N. Pavesic, "Adaptation of sift features for face recognition under varying illumination," in *Proc. MIPRO 2010*, pp. 691–694.

[11] D. P. Bertsekas, "The auction algorithm: A distributed relaxation method for the assignment problem," *Ann. Oper. Res.*, vol. 14, no. 1-4, pp. 105–123, 1988.

[12] "IBM System G," http://systemg.ibm.com/.

[13] J. Choi, J. J. Dongarra, L. S. Ostrouchov, A. P. Petitet, D. W. Walker, and R. C. Whaley, "Design and implementation of the scalapack lu, qr, and cholesky factorization routines," *Sci. Program.*, pp. 173–184, 1996.

[14] Robert D. Blumofe and Charles E. Leiserson, "Scheduling multithreaded computations by work stealing," *J. ACM*, vol. 46, no. 5, pp. 720–748, 1999.

[15] "ISVision Dataset," http://www.icme2014.org/isvision-challenge/.