Deterministic Load Balancing and Dictionaries in the Parallel Disk Model

Mette Berger, Esben Rune Hansen, Rasmus Pagh, Mihai Pătraşcu, Milan Ružić, and Peter Tiedemann
The Dictionary Problem

How to store a set \( S \subset U \) and support inquires about membership – “Is \( x \in S \)?” – plus retrieval of associated data.

Static vs. dynamic.

Some concepts that have given good solutions: hashing, trees.

Further specifications:

- Finite universe \( U \) with elements of equal size.
- Space usage should be linear.
- Algorithms should be deterministic.
Parallel Disk Model

- $D$ storage devices.
- Memory on each device is seen as an array of memory blocks.
- Each block has capacity to store $B$ data items.
- One parallel I/O operation consists of retrieving (or writing) a block of memory from (or to) each of the $D$ storage devices.
- The performance of an algorithm is measured in the number of parallel I/Os it makes.
- We assume “small” amounts of available internal memory.
Cost of operations is $\Theta(\log_{BD} n)$ I/Os in the worst case, which means that no asymptotic speedup is achieved compared to the one disk case unless the number of disks is very large, $D = B^{\omega(1)}$.

The search procedure makes $\Theta(\log_{BD} n)$ adaptive probes.

Supports a wider range of operations.

It does not perform well in a case of long or variable-length keys, and our structure is similar in this respect.
## Performance of Hashing Schemes

<table>
<thead>
<tr>
<th>Method</th>
<th>Lookup I/Os</th>
<th>Update I/Os</th>
<th>Bandwidth</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>DGMP92</td>
<td>$O(1)$</td>
<td>$O(1)$ whp.</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Hashing, no overflow</td>
<td>1 whp.</td>
<td>2 whp.</td>
<td>$O\left(\frac{BD}{\log n}\right)$</td>
<td>$BD = \Omega(\log n)$</td>
</tr>
<tr>
<td>Cuckoo hashing</td>
<td>1</td>
<td>$O(1)$ am. exp.</td>
<td>$O(BD)$</td>
<td>-</td>
</tr>
<tr>
<td>DGMP92 + trick</td>
<td>$1 + \epsilon$ avg. whp.</td>
<td>$2 + \epsilon$ avg. whp.</td>
<td>$O(BD)$</td>
<td>-</td>
</tr>
</tbody>
</table>

All hashing based dictionaries (we are aware of) may use $n/B^{O(1)}$ I/Os for a single operation in the worst case.
Our algorithms assume access to certain expander graphs “for free”.

The graphs that we use are bipartite. In a bipartite graph $G = (U, V, E)$, we may refer to $U$ as the “left” part, and refer to $V$ as the “right” part.

**Definition.** A bipartite, left-d-regular graph $G = (U, V, E)$ is a $(d, \varepsilon, \delta)$-expander if any set $S \subset U$ has at least
\[
\min ((1 - \varepsilon)d|S|, (1 - \delta)|V|) \text{ neighbors.}
\]

**Notation:** $u = |U|$, $v = |V|$, $n = |S|$.
There exist \((d, \varepsilon, \delta)\)-expanders with left degree
d \(= O(\log(\frac{u}{v}))\), for any \(v\) and positive constants \(\varepsilon, \delta\).

For applications one needs an explicit expander, i.e. an expander for which we can efficiently compute the neighbor set of a given node (in the left part).

No explicit constructions with the mentioned (optimal) degree are known. The currently best explicit construction has degree of \(2^{(\log \log u)^{O(1)}}\).

We assume availability of an explicit striped expander graph of degree \(O(\log u)\).
Our Results vs. Hashing

<table>
<thead>
<tr>
<th>Method</th>
<th>Lookup I/Os</th>
<th>Update I/Os</th>
<th>Bandwidth</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>DGMP92</td>
<td>$O(1)$</td>
<td>$O(1)$ whp.</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>A</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>-</td>
<td>$D = \Omega(\log u)$</td>
</tr>
<tr>
<td>Hashing, no overflow</td>
<td>1 whp.</td>
<td>2 whp.</td>
<td>$O\left(\frac{BD}{\log n}\right)$</td>
<td>$BD = \Omega(\log n)$</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>2</td>
<td>$O\left(\frac{BD}{\log n}\right)$</td>
<td>$D = \Omega(\log u)$, $B = \Omega(\log n)$</td>
</tr>
<tr>
<td>Cuckoo hashing</td>
<td>1</td>
<td>$O(1)$ am. exp.</td>
<td>$O(BD)$</td>
<td>-</td>
</tr>
<tr>
<td>DGMP92 + trick</td>
<td>$1 + \epsilon$ avg. whp.</td>
<td>$2 + \epsilon$ avg. whp.</td>
<td>$O(BD)$</td>
<td>-</td>
</tr>
<tr>
<td>C</td>
<td>$1 + \epsilon$ avg.</td>
<td>$2 + \epsilon$ avg.</td>
<td>$O(BD)$</td>
<td>$D = \Omega(\log u)$, $B = \Omega(\log n)$</td>
</tr>
</tbody>
</table>
Framework for Our Dictionaries

- It is sufficient to describe structures that support only lookups and insertions into a set whose size is not allowed to go beyond $N$, where the value of $N$ is specified on initialization of the structure.

- The dictionary problem is a *decomposable search problem*, so we can apply standard, worst-case efficient global rebuilding techniques to get fully dynamic dictionaries, without an upper bound on the size of the key set, and with support for deletions.

- The amount of space used and the number of disks increase by a constant factor compared to the basic structure.
In general setting: a set of servers and a set of (possible) clients. For each client there are permissible servers that can execute the client’s jobs. The problem is to assign jobs of given clients in a way that optimizes some objective – e.g. $L_p$ norm of latencies.

- Versions with servers having different speeds or equal speeds.
- In an online algorithm, jobs are revealed for one client at a time, and the algorithm has to decide how to distribute the jobs without knowing the future requests by clients.
- The clients-servers relation can be represented with a bipartite graph.
We analyzed a natural load balancing scheme for a \((d, \varepsilon, \delta)\) expander graph.

Our problem setting: suppose there is an unknown set of \(n\) left vertices where each vertex has \(k\) items, and each item must be assigned to one of the neighboring right vertices (called “buckets”); the problem is to balance \(L_1\) norm) the numbers of items assigned to buckets in online fashion.

A natural greedy strategy is this: assign the \(k\) items of a vertex one by one, putting each item in a bucket that currently has the fewest items assigned, breaking ties arbitrarily.

We showed that the maximum load that the greedy scheme makes is close to the average load of \(kn/v\).
Lemma If $d > \frac{k}{1-\epsilon}$ then after running the load balancing scheme using a $(d, \epsilon, \delta)$-expander, the maximum number of items in any bucket is bounded by $\frac{kn}{(1-\delta)v} + \log(1-\epsilon)\frac{d}{k} v$.

Proof idea. Let $B(j)$ denote the number of buckets having more than $j$ items. Bounds on values $B\left(\frac{kn}{(1-\delta)v} + i\right)$ are determined by an induction on $i$ and using the properties of the expander graph.
Dictionary Construction

- Use a striped expander graph $G$ and an array of $v$ (more elementary) dictionaries.
- The array is split across $D = d$ disks according to the stripes of $G$.
- The dictionary implements the load balancing scheme with parameters $v, k$ set to ensure a load of size $\Theta(\log N)$ on each bucket.
- For the result A set $k = 1$ and $v = N / \log N$.
- For the result B set $k = d/2$ and $v = kN / \log N$. 
We first consider static assignments.

- **Lemma.** We call *unique neighbor* nodes those elements of $\Gamma(S)$ that have exactly one member of $S$ as a neighbor.

- **Lemma.** There are “many” unique neighbor nodes.

- **Lemma.** There are “many” elements of $S$ having “many” unique neighbors.

These observations lead to an almost optimal static dictionary. The static dictionary can be dynamized, then giving the result C.
An expander construction is considered explicit if one can evaluate $\Gamma(x)$ in time $\text{polylog}(u)$, given $u$ and $v$.

It is not known how to obtain an optimal explicit expander, when $u = \omega(N)$.

In the context of the external memory model it makes sense to allow an expander construction to make use of a small amount of internal memory, as long as no access to the external memory is made on evaluation of $\Gamma(x)$.

We consider *semi-explicit* expander constructions, which use $o(N)$ words of internal memory and are allowed a pre-processing step, but evaluation of $\Gamma$ still takes time $\text{polylog}(u)$ with no access to external memory.
We attained a new construction of an unbalanced expander, when $u$ is polynomial in $N$.

Our construction requires a degree of $O((\log N)^{O(1)})$, compared to the previous best result of $O((\log \log N)^2)$. 

The method requires $O(N^\beta)$ words of internal memory, for any constant $\beta > 0$.

Still not a striped expander.