Cascade Hypothesis for Dynamic Balanced Graph Partitioning

x
y

1 Problem Definition of Dynamic Balanced Graph Partitioning

Nodes and clusters. There is a set of n nodes, initially distributed arbitrarily across ℓ clusters, each of size k. We call two nodes $u, v \in V$ *collocated* if they are in the same cluster.

Input sequence. An input to the problem is a sequence of communication requests $\sigma = (u_1, v_1)$, (u_2, v_2), $(u_3, v_3), \ldots$, where pair (u_t, v_t) means that the nodes u_t, v_t exchange a fixed amount of data. At any time t, an algorithm needs to serve the communication request (u_t, v_t).

Reconfigurations. Right before serving the request, the algorithm can move nodes between clusters. The nodes fit perfectly into the clusters, i.e., $n = k \cdot \ell$. Due to cluster capacity constraints, a node can never be migrated alone, but it must be *swapped* with another node at a cost of $2 \cdot \alpha$. We also assume that when an algorithm wants to migrate more than two nodes, this has to be done using several swaps, each involving two nodes.

The cost. We assume that a communication request between two collocated nodes (i.e., placed on the same server) costs 0. The cost of a communication request between two nodes located in different clusters is 1, and the cost of migrating a node from one cluster to another is $\alpha \geq 1$, where α is a parameter (an integer). The total cost of an algorithm ALG, consisting of communication plus migration cost.

1.1 Online Algorithms and Competitive Analysis

The input to the problem is revealed one-by-one, in online fashion. Upon seeing a request, the algorithm must serve it without the knowledge of future requests (although prior to serving the request, it can reconfigure the nodes).

We measure the performance of an online algorithm by comparing to the performance of an optimal offline algorithm. Formally, let $\mathsf{ONL}(\sigma)$, resp. $\mathsf{OPT}(\sigma)$, be the cost incurred by an online algorithm ONL, resp. by an optimal offline algorithm OPT, for a given sequence of requests σ. In contrast to ONL, which learns the requests one-by-one as it serves them, OPT has complete knowledge of the entire request sequence σ *ahead of time*. The goal is to design online repartitioning algorithms that provide worst-case guarantees. In particular, ONL is said to be *ρ-competitive* if there is a constant β, such that for any input sequence σ it holds that

$$\mathsf{ONL}(\sigma) \leq \rho \cdot \mathsf{OPT}(\sigma) + \beta.$$

Note that β cannot depend on input σ but can depend on other parameters of the problem, such as the number of nodes or the number of clusters. The minimum ρ for which ONL is ρ-competitive is called the *competitive ratio* of ONL.

Note on running time. Although it is vital to keep the running time of an online algorithm low, it is not a strict requirement. In contrast to exact algorithms or approximation algorithms, the running time of an online algorithm may be exponential. The algorithmic challenge lies in making decisions without the knowledge of the future.

2 The algorithm CAS

We introduce $O(k^2 \cdot \ell^2)$-competitive deterministic algorithm CAS. At any time, CAS serves a request, adjusts its internal structures (defined below) accordingly and then possibly migrates some nodes. CAS operates in phases, and each phase is analyzed separately. The first phase starts with the first request.

In a single phase, CAS maintains a helper structure: a complete graph on all $\ell \cdot k$ nodes, with an edge present between each pair of nodes. We say that a communication request is *paid* (by CAS) if it occurs between nodes from different clusters, and thus entails a cost for CAS. For each edge between nodes x and y, we define its weight $w(x, y)$ to be the number of paid communication requests between x and y since the beginning of the current phase.

Whenever an edge weight reaches α, it is called *saturated*. If a request causes the corresponding edge to become saturated, CAS computes a new placement of nodes (potentially for all of them), so that all saturated edges are inside clusters (there is only one new saturated edge). Among all possible placements, CAS chooses the closest to the current configuration. If finding such placement is not possible, node positions are not changed, the current phase ends with the current request, and a new phase begins with the next request. Note that all edge weights are reset to zero at the beginning of a phase.

▶ **Theorem 1.** *CAS is $O(k^2 \cdot \ell^2)$-competitive.*

Proof. We bound the costs of CAS and OPT in a single phase. First, observe that whenever an edge weight reaches α, its endpoint nodes will be collocated until the end of the phase, and therefore its weight is not incremented anymore. Hence the weight of any edge is at most α.

Second, observe that the graph induced by saturated edges always constitutes a forest. Suppose that, at a time t, two nodes x and y, which are not connected by a saturated edge, become connected by a path of saturated edges. From that time onward, CAS stores them in a single cluster. Hence, the weight $w(x, y)$ cannot increase at subsequent time points, and (x, y) may not become saturated. The forest property implies that the number of saturated edges is smaller than $k \cdot \ell$.

The two observations above allow us to bound the cost of CAS in a single phase. The number of reorganizations is at most the number of saturated edges, i.e., at most $k \cdot \ell$. As the cost associated with a single reorganization is $O(k \cdot \ell \cdot \alpha)$, the total cost of all node migrations in a single phase is at most $O(k^2 \cdot \ell^2 \cdot \alpha)$. The communication cost itself is equal to the total weight of all edges, and by the first observation, it is at most $\binom{k \cdot \ell}{2} \cdot \alpha < k^2 \cdot \ell^2 \cdot \alpha$. Hence, for any phase P (also for the last one), it holds that $\mathsf{CAS}(P) = O(k^2 \cdot \ell^2 \cdot \alpha)$.

Now we lower-bound the cost of OPT on any phase P but the last one. If OPT performs a node swap in P, it pays $2 \cdot \alpha$. Otherwise its assignment of nodes to clusters is fixed throughout P. Recall that at the end of P, CAS failed to reorganize the nodes. This means that for any static mapping of the nodes to clusters (in particular the one chosen by OPT), there is a saturated inter-cluster edge. The communication cost over such an edge incurred by OPT is at least α (it can be also strictly greater than α as the edge weight only counts the communication requests paid by CAS).

Therefore, the CAS-to-OPT cost ratio in any phase but the last one is at most $O(k^2 \cdot \ell^2)$ and the cost of CAS on the last phase is at most $O(k^2 \cdot \ell^2 \cdot \alpha)$. Hence, $\mathsf{CAS}(\sigma) \leq O(k^2 \cdot \ell^2) \cdot \mathsf{OPT}(\sigma) + O(k^2 \cdot \ell^2 \cdot \alpha)$ for any input σ. ◀

3 The Cascade Hypothesis.

Upon receiving a request, CAS may merge two of its components and consequently perform a reconfiguration of its nodes. The cascade hypothesis aims to bound the cost of such reconfiguration.

In [ALPS16], authors trivially bounded the reconfiguration cost for each merge action by $k \cdot \ell$ (the proof is given also in this manuscript in Theorem 1). This roughly corresponds to migrating every node in every cluster. In [PPS], authors significantly improved the analysis of CAS for unsaturated edges (that in principle incur the cost $O(k^2 \cdot \ell^2)$). This allows to bound the ratio by $O(k \cdot \ell \cdot f(k, \ell))$, where $f(k, \ell)$ is a bound on the cost of a single repartition. The best known lower bound is $\Omega(k \cdot \ell)$ [PPS].

In this manuscript, we state two hypotheses. Proving any of them brings an immediate improvement to the competitive ratio of CAS.

▶ **Hypothesis 1.** ***Strong cascade hypothesis.*** *The cost of a single reconfiguration of CAS can be bounded by f that does not depend on ℓ.*

▶ **Hypothesis 2.** ***Weak cascade hypothesis.*** *The cost of a single reconfiguraton of CAS can be bounded by f that is $o(\ell)$ (i.e., the dependency on ℓ is sublinear).*

In both hypotheses, the cost of reconfiguration may be exponential in k. One can think that k is usually small in comparison to ℓ.

3.1 Characterization of the migration graph

Consider a single component merge action performed by CAS that triggers a reconfiguration from a configuration C_I to a configuration C_F. To perform the reconfiguration, the nodes of two non-collocated components migrate to a common cluster. Each cluster contains exactly k nodes, thus other nodes migrate to take the place of nodes of merged components. The place they leave must be filled by other nodes, thus a single merge action may trigger migrations in clusters not directly involved in a merge action. Both C_I and C_F are component-respecting configurations (i.e., saturated edges are kept inside clusters), hence a migration of a single node entails migrations of all nodes of its cluster.

To help analyzing the cost of a reconfiguration, we introduce a *migration graph* that models the reconfiguration. In the following, we characterize the structure of the migration graph.

Vertices of the migration graph and the core vertices. The vertices of the migration graph are all ℓ clusters of the instance. We distinguish the *core vertices* that correspond to clusters directly involved in the merge operation: the clusters containing the to-be-merged components in C_I and the cluster containing the merged component in C_F. There are at most 3 core vertices in each migration graph (at most two components participate in the merge, and these may migrate to a third cluster).

Edges and their labels. The edges of the migration graph denote the migration of nodes from C_I to C_F. Each edge is labeled with a set of components that migrate between clusters. The components are indistinguishable, hence the label is a multiset of component sizes. The *weight* of an edge is the sum of sizes of components of its label. Each edge is directed from the cluster that contained the nodes to the cluster they migrated to. Both edges might exists between a pair of nodes, as whole components of nodes migrate, and the back-and-forth exchange may be needed for the configuration C_F to be component-respecting.

Vertex degree. At most k nodes may migrate from a cluster and at most k nodes may migrate into a cluster. Thus, the sum of the labels for both ingoing and outgoing edges of each vertex is at most k. This implies that the number of ingoing and outgoing edges is also bounded by k.
Flow preservation. In any configuration (including C_I and C_F), each cluster contains exactly k nodes. Thus, for any vertex, the sum of labels of ingoing edges must equal the sum of labels of outgoing edges.
Migration graph and the cost of reconfiguration. The cost of cluster reconfiguration equals the number of exchanged nodes multiplied by α. From the standpoint of a migration graph, this corresponds to the sum of labels on edges of the graph. A trivial upper bound on the cost is the size of the connected component in the migration graph multiplied by $2\alpha k$.

Note that for a single component merge, multiple migration graphs may exist (it depends on the choice of the algorithm), and multiple of them might have the optimal cost.

4 Tasks

The main task is to prove weak and strong cascade hypotheses or provide a counterexample. In this section we propose a number of programming tasks. Their main objective is to help understanding the problem. Moreover, the tools may be used to quickcheck if the cascade graph has certain properties.

4.1 Minimum cost cascade.

We are given integers k, ℓ, an initial configuration C_I of clusters (the sizes of components in each cluster), and distinguished components A and B that are present in the initial configuration. We say that a configuration is feasible if each component is entirely contained in a single cluster (i.e., no component is split), and each cluster contains exactly k nodes.

The objective is to perform a merge operation of A and B. Precisely,

1. Determine if there exists a feasible configuration of clusters after the merge.
2. Find a feasible configuration that is closest to C_I (the number of node migrations is minimized).
3. If multiple configurations with minimum cost exists, find all of them.

Implementation. Use existing tools such as Mixed-Integer Programming solvers, CP-SAT solvers or other libraries, specialized in searching spaces of exponential size. Do not implement your own solutions unless justified.
The solution must be exponential due to NP-completeness of ℓ-way integer partition [AR06].

4.2 Unique minimum cost cascade.

Multiple min-cost cascades exists, and many of them may be isomorphic. Propose a simple rule to determine a unique cascade (e.g. the cascade with components of minimum index, where index is a component ID). The rule should be easy to express mathematically and easy to implement.

4.3 Cascade graph analysis.

Fix a cascade graph, and let C_F be the final configuration. The *core* vertices of the cascade graph are: the cluster that contains A in C_I, the cluster that contains B in C_I, and the cluster that contains $A \cup B$ in C_F.

Determine the following properties of the given cascade graph:

1. The length of the longest simple cycle (i.e., without repeating vertices).
2. Existence of ears: vertices that do not belong to any simple cycle that contains a core vertex.

4.4 Generating initial configurations.

Automate graph analysis by generating feasible initial configurations. Two possible solutions:

1. Use random feasible configurations.
2. Enumerate over all possible non-isomorphic configurations for a given small k and ℓ.

References

ALPS16 Chen Avin, Andreas Loukas, Maciej Pacut, and Stefan Schmid. Online Balanced Repartitioning. *DISC*, pages 243–256, 2016.

AR06 Konstantin Andreev and Harald Räcke. Balanced graph partitioning. *Theory of Computing Systems*, 39(6):929–939, 2006.

PPS Maciej Pacut, Mahmoud Parham, and Stefan Schmid. Optimal algorithms for dynamic balanced graph partitioning. In *unpublished*.