Getafix: Workload-aware Data Management in Lookback Processing Systems

Presenter: Mainak Ghosh
Collaborators: Thomas Kao, Le Xu, Indranil Gupta

Lookback Processing System (LPS)

Interactive systems:

- Stream Processing Systems -- Naiad, Spark Streaming
- Data Warehouses -- Dremel, Impala

Lookback Processing Systems -- Warehouse for time series data. -- Druid, Trill

Data stored in batches (*segments*) -- e.g. hours worth of click logs

- Segments are *immutable*

Queries are *aggregation* (sum, count, etc.) over a time period's worth of collected data

- Queries access multiple segments. Each *segment query pair* can be scheduled in parallel

Motivation

Segment based processing raises two questions:

- Which segments should be loaded?
- How many replicas to assign each?

Current LPS decouple segment management and query routing

- Minimize latency and improve throughput

LPS uses workload assumptions

- Recent segments assigned to “hot tier” -- larger replication

Problems:

Contribution

We propose segment management strategies which use segment popularity from the queries

- For static workloads, we show that our strategies are optimal
- For dynamic workloads, we use segment popularity history

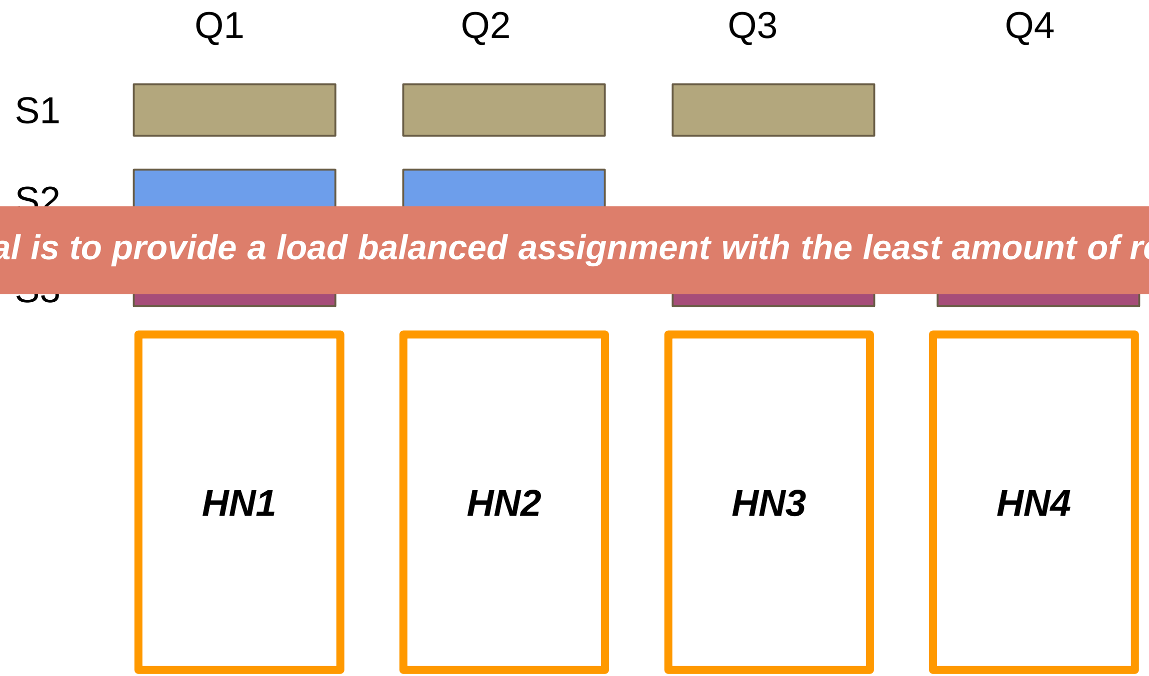
Through simulation, we show that our strategies successfully reduce replication between 20 - 30% with minimal impact on system throughput

Future Work

Design and implement Getafix on top of Druid

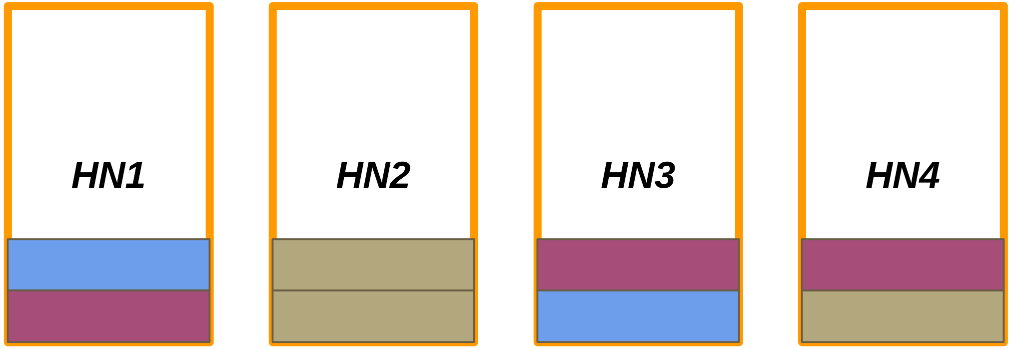
Evaluate the performance of Getafix with real-world workload traces from Yahoo

Problem



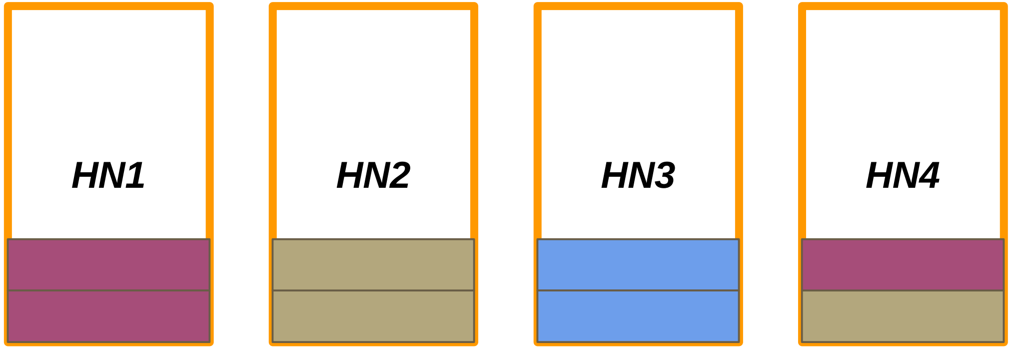
Goal is to provide a load balanced assignment with the least amount of replication

A possible solution...



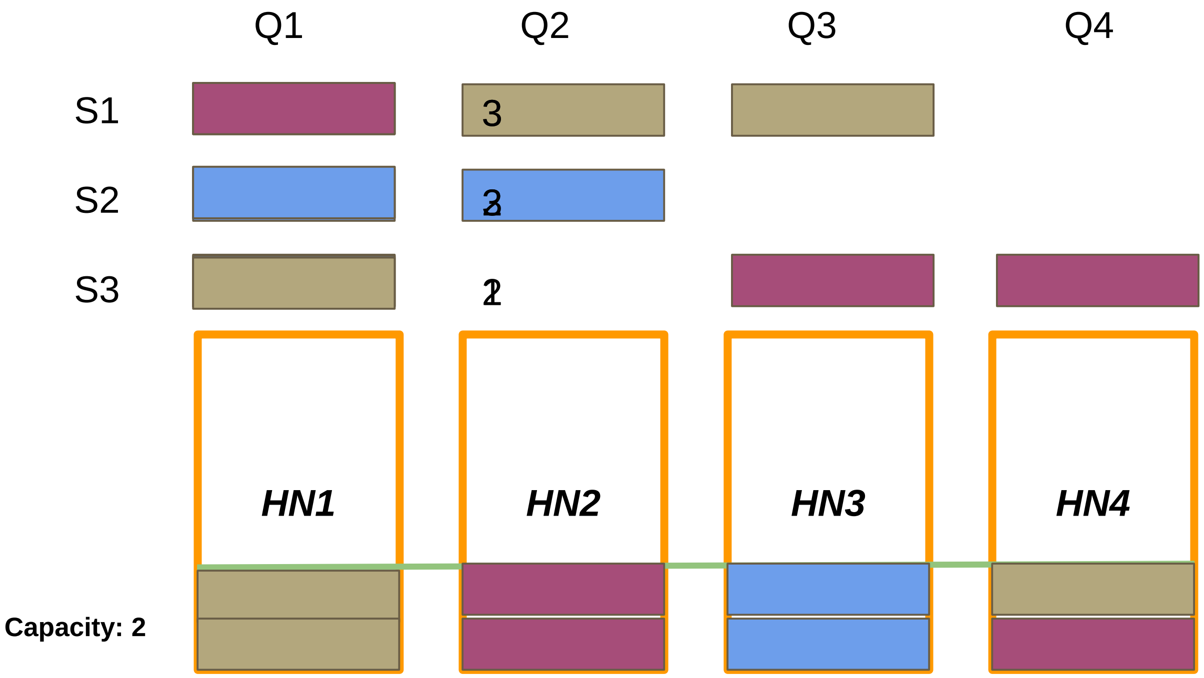
Load Balanced Assignment. Number of Replicas: 7

Correct Solution



Load Balanced Assignment. Number of Replicas: 5

Algorithm



Assign bin capacity as total number of query segment pairs by number of historical nodes

Create a priority queue on segments by access count

Pick the highest priority segment, assign to historical node based on a policy and return the rest to the priority queue

Continue till the queue is not empty

Policy

First Fit: Choose the first HN that is not yet full.

Large

Best Fit Policy provides a load balanced assignment with the least amount of replication

Best Fit: Choose the HN which would have the least space remaining after accommodating all the queries in the current segment.

- If none of the nodes have sufficient capacity, the node with the largest available capacity is chosen, filled and the remaining queries are returned to the queue.

Dynamic Solution

In a dynamic system, we break up the execution over multiple time windows

Segment accesses are counted in a time window

Popularity of a segment is the weighted average of its access counts over a fixed number of past windows

Exponentially decay past window counts

Run the best fit algorithm which returns the expected number of segment replicas in the system

Segments are loaded if the expected count is larger than current

Otherwise, segments are removed

Simulator Model

Queries are characterized by 1) start time and 2) query size (number of segments)

Each value is selected using a distribution:

- Start Time: 1) Latest, 2) Uniform and 3) Zipfian
- Query Size: 1) Zipfian, 2) Uniform

(Latest, Zipfian) implies recent segments are more popular and small query sizes (few hours) are more popular than larger ones (few days). We use this as default.

Segment Loader: We use Druid's Cost based greedy strategy

Query Router: A segment query pair is routed to the HN which has the least query load

Simulator Model

Segment Management Strategies:

- Fixed: All segments are replicated fixed number of times
- Tiered: Segments are divided into tiers based on age. Recent segments are assigned to hot tier. Warm tier houses older segment. Very old segments are removed assuming obsolete. We use hot tier threshold of 300 and warm tier threshold of 800.
- Adaptive: In this strategy, segment count information is collected from router. Number of replicas are calculated as $c_i * n / \Sigma c_i$
- Best Fit: Described earlier

Simulation

Settings: Experiment runs for 1000 time units. 1 segment and 6 queries injected per time unit. Number of HN is 10. Power law constant 1.2.

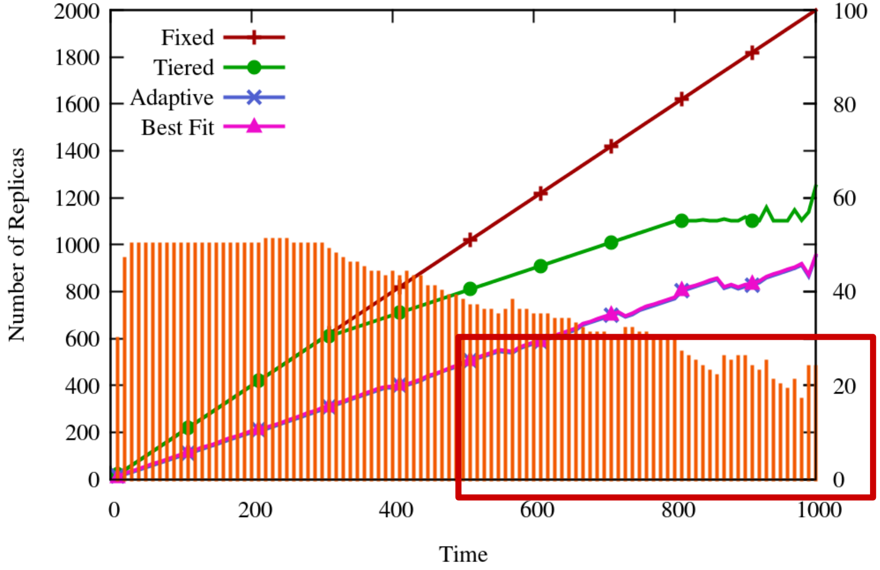
Scenarios: Outside the normal, we also implemented some other common scenarios:

- Varying Workload
- Bursty Segment
- Bursty Query

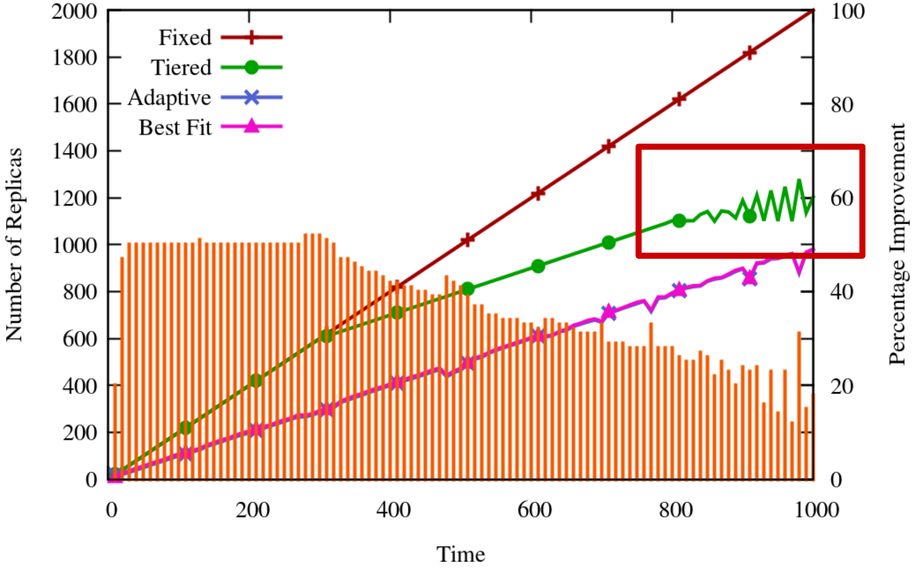
Metrics: We measure performance using:

- Total Replication

Comparison -- Number of Replicas

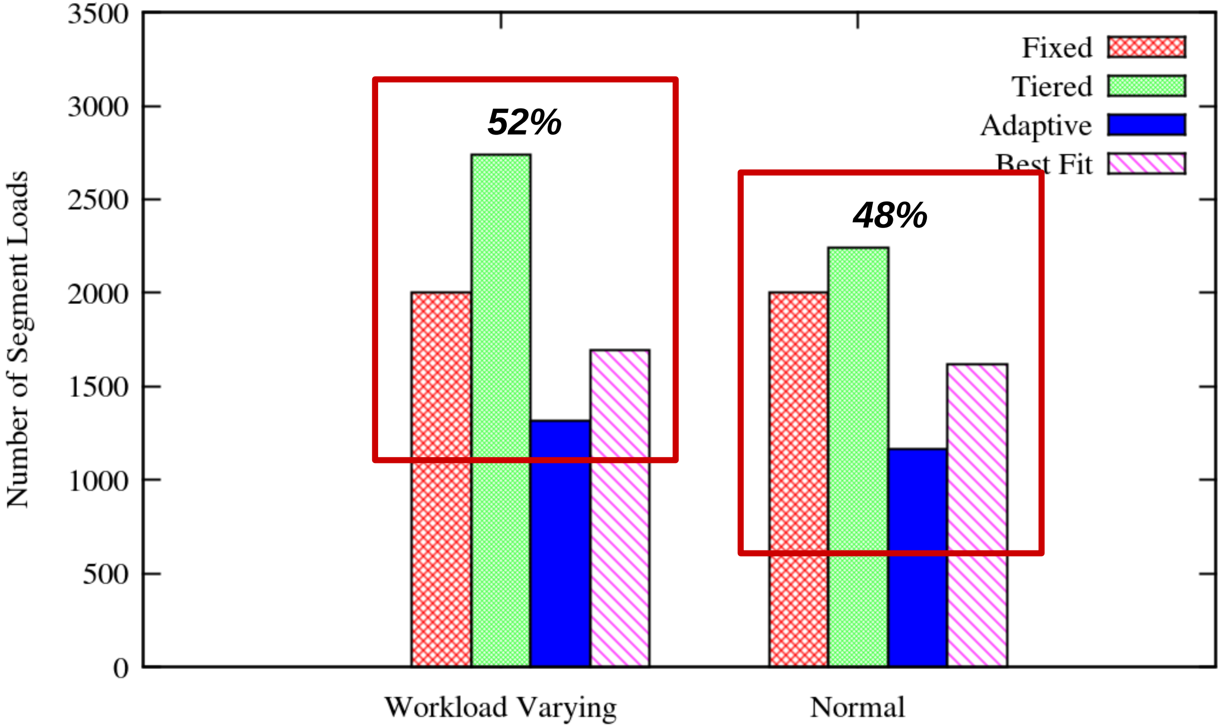


Normal, Latest, Zipfian

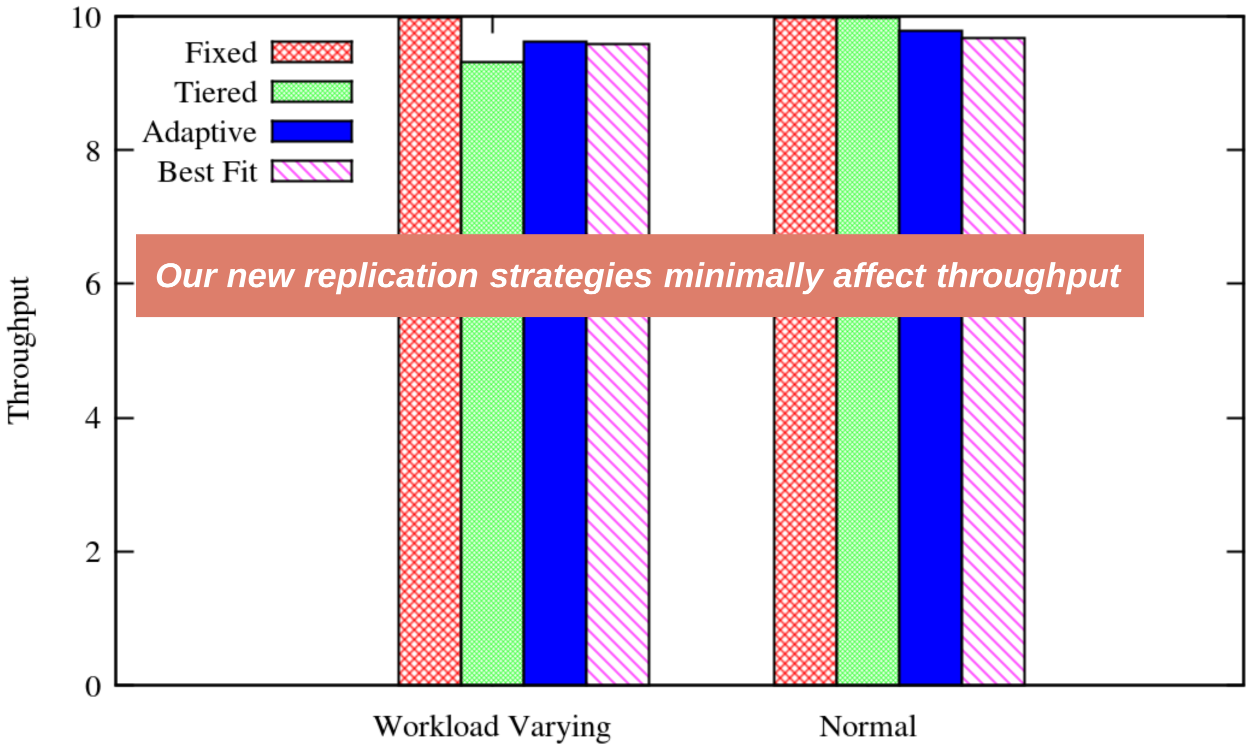


Workload Varying, Latest, Zipfian

Comparison -- Number of Segment Loads

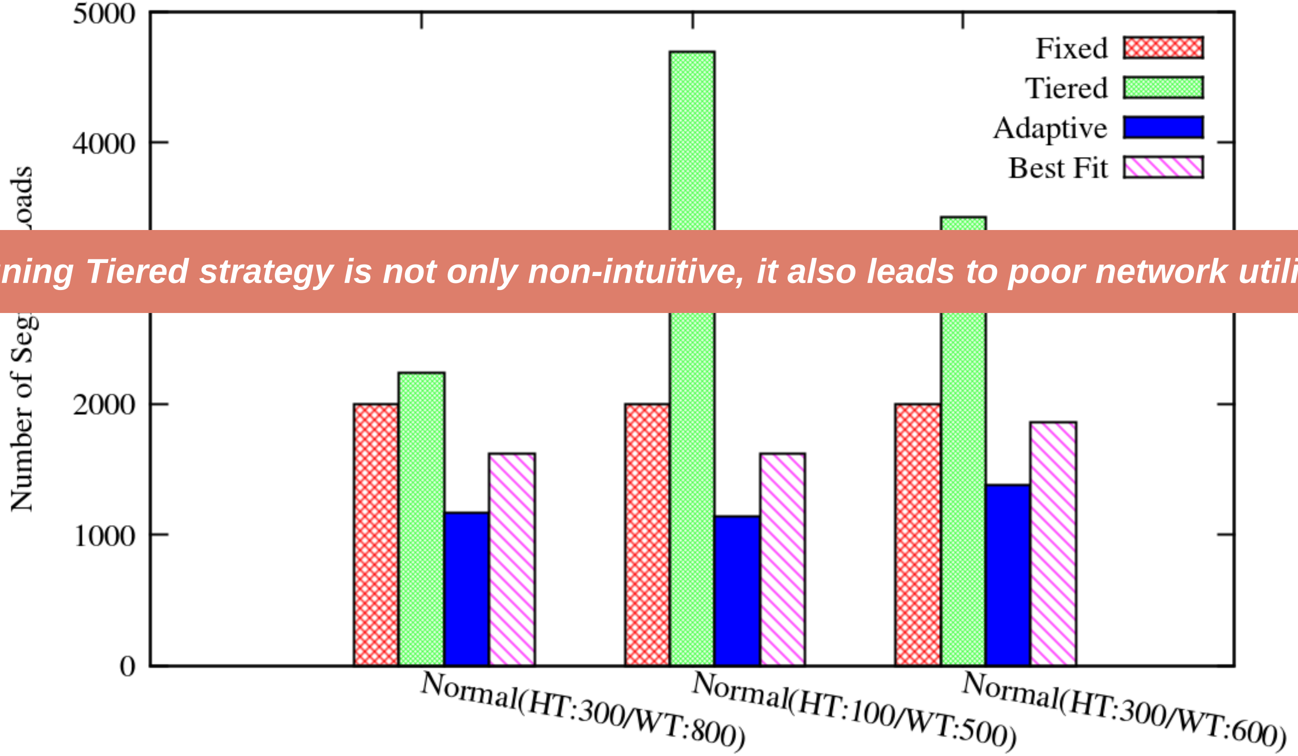


Comparison -- Throughput



Our new replication strategies minimally affect throughput

Tuning Tiered



Hand tuning Tiered strategy is not only non-intuitive, it also leads to poor network utilization

Related Work

Workload aware data partitioning was explored in Schism [Curino et. al.]

MeT [Cruz et. al.] solves a similar problem in NoSQL databases

Data aware task placement has been explored in clustering and computing frameworks

- Ours is the first work to handle popularity aware data replication in LPS

Adaptive schemes have been used for replicating read/write objects to improve operation latency [Wolfson et. al.]

It has also been used to reduce storage and network overhead in companies like Facebook [Muralidhar et. al.]

Summary

We proposed techniques for segment management in lookback processing systems

We use segment popularity information to decide which segments to load and how many replicas to assign

Best Fit policy is optimal in a static setting

In dynamic setting, Best Fit and Adaptive strategies improve storage utilization by 20% and network utilization by 48%

Thank you

Problem

We assume each query takes same amount of time (unit time) to process a segment it touches.

Uniformly distributing these segment query pairs will give time optimal schedule

May not be the least amount of replication

Similar to *bin packing problem*

In this problem, we have to place segment query pair (ball) to HN (bin) such that the sum of the number of unique segments in each bin is minimized

Algorithm

input: C: Access counts for each segment
$nodelist$: List of HNs
Algorithm MODIFIEDFIT(C, $nodelist$)
$n \leftarrow$ LENGTH($nodelist$)
$capacity \leftarrow \frac{\sum_{C_i \in C} |C_i|}{n}$
$binArray \leftarrow$ ALLOCATE($n, capacity$)
$priorityQueue \leftarrow$ BUILDMAXHEAP(C)
while !EMPTY($priorityQueue$) **do**
$(segment, count) \leftarrow$
EXTRACT($priorityQueue$)
$(left, bin) \leftarrow$ CHOOSEHISTORICALNODE
$(count, binArray)$
LOADSEGMENT($nodelist, bin, segment$)
if $left > 0$ **then**
INSERT($priorityQueue, (segment, left)$)
end
end

Algorithm 1: Generalized Allocation Algorithm.