SimPl: A Simulation Platform for Elastic Load-Balancing in a Distributed Spatial Cache Overlay

Carlos Lübbe, Nazario Cipriani
Universität Stuttgart, Institute of Parallel and Distributed Systems
Universitätsstraße 38, 70596 Stuttgart, Germany
Email: {luebbe, cipriani}@ipvs.uni-stuttgart.de

Abstract—Location-based services (LBS) have to cope with increasing query loads at their data tier. Yet, the data access patterns of LBS typically possess spatial locality. Therefore, a dedicated spatial cache which provides efficient access to the data currently needed may considerably reduce this load. To ensure high throughput throughout the entire execution, previous work introduced an elastic load-balancing mechanism for multiple cache nodes that collaborate in a distributed spatial cache overlay. However, calibrating such a load-balancing mechanism is a non-trivial task, as several parameters influence such a system.

We demonstrate a simulation platform (SimPl) for elastic load-balancing. SimPl enables a network administrator to set up several overlay topologies and calibrate their system parameters using different spatial data access patterns. A live visualization of the simulated overlay enables intuitive comparison of overlay topologies and their load-balancing abilities.

I. INTRODUCTION

Smart mobile devices providing high-speed Internet access give users access to information anytime and anywhere. On that account, location based services (LBS) have gained enormous popularity, as they consider the spatial context of users which is particularly handy for daily life. In consequence, the data tier of an LBS has to cope with a huge amount of load. Yet, such load possesses high spatial and temporal locality, as relevant data is closely connected to behavioral patterns of users. Hence, a dedicated spatial cache which provides efficient access to the data currently needed may considerably alleviate the load. To guarantee high throughput, multiple cache nodes can collaborate in a distributed spatial cache overlay which balances the load among the nodes. In previous work [1], we introduced an elastic load-balancing mechanism for a distributed spatial cache overlay (DiSCO), which bases on the physical model of a particle-spring system. Using spring contraction, cache nodes instantly form processing clusters in regions with high load and thus adapt to rapidly shifting hot spots.

However, it is a non-trivial task for a network administrator to set up and calibrate such an overlay, as a multitude of environmental effects and system parameters have to be considered. For instance, a major challenge is to decide on the sensibility of the elastic load-balancing mechanism, as it determines the time the overlay needs to react to shifting load. Undervalued sensibility may reduce the effectiveness of load-balancing, whereas too high values cause overreaction. In addition, calibrating such parameters during execution is tedious, as it may take a long time until the effects of calibration can be observed. Moreover, it may not be wise to fiddle with parameters of a production system during execution, as disadvantageous tuning may degrade the overall performance. Thus, means to calibrate system parameters off-line are required.

In this demonstration, we present a simulation platform (SimPl) for elastic load-balancing in a DiSCO. Our framework enables the network administrator to determine feasible system parameters and configure a DiSCO prior to execution as sketched in Figure 1. Properly configured, the elastic load-balancing mechanism automatically adjusts to shifting load without any intervention of the network administrator and thus ensures high throughput throughout the entire execution. In detail, SimPl enables the network administrator to:

- include and compare user defined overlay topologies,
- examine the impact of different spatial and temporal data access patterns on the cache performance,
- calibrate system parameters during simulation,
- instantly observe calibration effects through a live visualization of the overlay network and
- switch between different simulation views that visualize aspects of the current system state.
II. ELASTIC LOAD-BALANCING IN A DISCO

In previous work [1], we introduced an elastic load-balancing mechanism for a distributed spatial cache overlay (Disco) of cache nodes. In a Disco each node keeps an instance of a dedicated spatial cache which can store data of previous range queries. To manage spatial data, the cache bases upon a uniform grid. Each grid cell corresponds to a spatial region and keeps objects intersecting that region. A Disco combines these caches to form a distributed network of cache nodes. Key challenges for such an overlay is to cooperatively control the overall cache content, to balance the load among the nodes and to maintain an overlay topology which enables efficient data access. We detail on these aspects in the following.

A. Controlling the cache content

To control which data should be currently kept by a node, we use a dedicated cell replacement strategy, once the capacity of the cache is reached. Exploiting the spatial ordering of our cache, we replace the cells which have highest Euclidean distance to a certain point in space, denoted as cache focus. As a node will rather cache objects near its cache focus, a node’s preferred region for caching can be controlled by updating its cache focus. For instance, Figure 2(a) highlights the spatial regions currently cached by nodes which are a long way away from each other. Increasing the density of nodes in a spatial region also increases the probability of data being cached by some node in this region, as shown in Figure 2(b). Following this observation, it is our goal to position the cache focuses of participating nodes in such a way that node density increases in regions with high load and decreases in low load regions.

B. Elastic Load-Balancing

In order to position the nodes in space, we borrow from the physical model of a particle-spring system. The model bases upon particles which are interconnected by springs. In the physical model the spring constant \( k \) resembles the springs strength and \( s \) its extension. According to Hooke’s law the spring’s force \( F \) is proportional to its strength and extension, i.e., \( F = k \cdot s \). We transfer this model to our distributed cache overlay: The nodes represent the particles and their cache focuses define their position. Neighboring nodes \( N_i \) of a node \( N \) are connected by springs and the Euclidean distance of their cache focuses define the springs extension \( s = |N_i - N|^2 \).

To achieve our goal – i.e., to increase node density in regions with high load and decrease it in low load regions – we contract the springs in high load regions and relax them in other regions. Most commonly, load is conceived as a measure for the amount of work a node has to do in a certain time period, such as queries per second. On the contrary, data-centered peer-to-peer systems often regard load as the amount of data which falls into a certain value range and thus into the responsibility of a certain node, e.g. [2]. Conceiving load as density of data can be quite beneficial in a system of distributed caches for spatial data. To consider both aspects (workload and density), we define a function denoted as gravity \( G \) of a node \( N \):

\[
G(N) = 1 + (\rho(N) \cdot \alpha + load(N) \cdot (1 - \alpha)) \cdot \beta
\]

Whereas, \( \rho(N) \) defines the density of cached data as number of cached objects divided by the number of occupied cells and \( load(N) \) defines workload of the node as queries per second. Weight \( 0 \leq \alpha \leq 1 \) is a parameter to adjust the relative importance between both aspects of load (i.e., density and workload). Sensibility \( 0 \leq \beta < \infty \) is a parameter to tune the influence of the gravity on the spring force. This parameter allows for adjusting the stiffness of the particle-spring system. A low value causes little contraction of springs, whereas a high value makes the system highly sensible to varying load.

Thus, for a spring \( i \) between a node \( N \) and its neighbor \( N_i \), the sum of their gravities defines the spring constant \( k_i = G(N) + G(N_i) \). The gravity causes an overloaded node to “pull” neighbors having less load closer to its own position. In Figure 2, the springs in the hot spot region contract causing the nodes to form a processing cluster of cache nodes in the hot spot region.

C. Building an elastic overlay topology

Using spring contraction to adapt the cache focuses in a load-aware manner, we need to ensure the physical stability of our particle-spring system. All unpinned particles, for instance, will eventually collapse into one point in space, as particles gravitate towards one another. Hence, we connect the border nodes to fixed anchor points, as shown in Figure 3. In addition, the nodes are connected to their neighboring nodes by links, through which they can send and receive messages and update status information about their neighbors. To find the most suitable node for processing a range query, the nodes exploit the topological relations...
between their cache focuses: A received query request is recursively forwarded to the neighbor having the smallest distance to the query region until no closer node is found. For instance, if node $A$ received a request with a query region close to node $I$, it would forward it to node $E$, which in turn would send it to node $I$.

### III. SimPl Architecture

SimPl packages numerous useful tools for simulating elastic load-balancing in a DiSCO. Figure 4 shows the general simulation process of SimPl.

The first tool, the **Query Generator**, creates a file containing range queries in sequential order, which represents the data access patterns of all clients. The **Simulator** is the main component of SimPl. It uses the previously generated query mix to simulate elastic load-balancing in a DiSCO. During execution the **Visualizer** component provides live views that enable the user to introspect the current state of the simulation. In addition, the Simulator writes detailed results into an output file using the widely supported comma separated values (csv) format. The output can be further processed by several **Post Processing Tools**. For instance, SimPl provides tools for data smoothing (e.g., moving average), for interpolation and others. All SimPl tools conform to the csv-format, so that alternative state of the art post processing tools or plotting programs such as gnuplot can be used. In the following, we detail on the Query Generator, the Simulator and the Visualizer.

#### A. Query Generator

The Query Generator creates a sequence of region queries that simulate the access patterns of clients. It models different client behaviors using spatial point distributions, whereas each point represents the center of a rectangular query region with configurable side length. The simplest distribution model generates a uniform distribution of queries within a specific region. As this decreases locality between successive queries, this model may be used as a worst case benchmark for caching. The Gaussian distribution model can be used to simulate hot spots centered at a fixed point in space with a configurable standard deviation that models the hot spots extent. Instead of using a fixed point as center of the hot spot, the user can also include trajectories representing a movement in time. By including such a trajectory, the moving Gaussian distribution model generates queries according to a Gaussian distribution whose center moves according to the trajectory points. The Query Generator outputs the queries into a file using the same format as for real queries. Thus, the Query Generator can be easily replaced by an online profiling component which enables including the query load of real clients into SimPl.

#### B. Simulator

The Simulator component bases on PeerSim [3], a discrete event-driven simulation framework for peer-to-peer networks. It models overlays as set of nodes, whereas each node contains a set of protocols that can send messages to other protocol instances on different nodes. As depicted in Figure 5, we implemented a stack of three protocols. The **Overlay Protocol** implements our elastic overlay topology. It is responsible for load-balancing, query routing and topology maintenance. In addition, it controls the cache focus of the **Cache Protocol** which implements the grid-based cache. The **Back-End Protocol** manages the connection to the data back-end.

At start the **Simulation Control** uses a **Topology Initializer** to create one protocol stack for each node and define their initial position in the overlay topology. As the Topology Initializer is a simple interface containing a single

![Figure 4. Simulation process of SimPl](image)

![Figure 5. Design of Simulator component](image)

![Figure 6. Visualizer component](image)
method, the user can easily include custom Initializers. After topology initialization, the Client Control sequentially sends queries to an Overlay Protocol instance of a randomly chosen node. The Overlay Protocol forwards the request to a node close to the query region. This node eventually processes the query by propagating it downwards the protocol stack to the Cache Protocol. The Cache Protocol probes the local cache and requests all missing parts of the query from the Data Source Protocol which fetches them from the data back-end. The Cache Protocol merges local cache results with back-end results and propagates the results upwards the protocol stack. The final results are sent back to the Client Control component. During execution, the Output Control writes simulation results to a file and propagates numerous details to the Visualizer component for live visualization. In addition, the Visualizer controls specific system parameters (sensibility and weight) during the simulation.

C. Visualizer

Figure 6 shows the graphical interface of the Visualizer component. On its top left side, the user can enable live views of the current simulation state. The Nodes view visualizes the nodes’ cache focuses as black circles and the springs as black lines. The Cache view shows the regions currently cached by the grid-based caches of all nodes as gray rectangles. Queries are depicted as red rectangles by the Queries view and the Data view draws the spatial outline of all objects stored in the data back-end as blue geometries. Using the sliders on the top right side, the user can adjust the system parameters (Sensibility and Weight) during simulation.

IV. DEMONSTRATION SHOWCASES

SimPl supports a network administrator to determine suitable system parameters for a DiSCO. Our live-demonstration includes typical showcases of this task:

Adjusting Sensibility: A key-performance parameter for elastic load-balancing is the sensibility of the underlying particle-spring system. Too low values lead to only marginal spring contraction which prevents effective load balancing (see Figure 7(a)). Too high values cause springs to contract rapidly which may destabilize and deform the particle-spring system (see Figure 7(b)). Through our visualization, the spectator can observe a particular interesting effect, which would have remained unnoticed without SimPl: For high sensibility values nodes between two neighboring hot spots oscillate being alternately attracted by their gravity.

Data density vs. Workload: Our system allows to weight the different aspects of load (data density or workload). On its extremes the system either reacts to changing workload and completely ignores data density or vice versa. Figure 7(c) shows an example for a data density prioritized simulation, where the distribution of nodes (black circles) roughly resembles the data distribution (blue geometries). Using the slider the spectator can intuitively adjust the weight and instantly observe the effects of his action. Thus, a compromise between workload-aware and density-aware adaption can be found.

Optional Showcases: Our demonstration includes several other showcases that can be presented on demand. These include: Determining the number of cache nodes, considering alternative data access patterns or including and examining alternative overlay topologies.

REFERENCES
