Functional Requirements

Erlang Router Team: Davis, George, Graham, Paul

Last Edit: December 9, 2014

Abstract

Abstract forthcoming

1 Introduction

The goal of this project is to develop a load-balancing system in an IOT (Internet of Things) environment. A multitude of devices with varying processing capabilities will be connected to a cluster of interconnected servers in a datacenter (a "cloud") via persistent TCP connections. These devices are known as "clients".

Clients are each assigned to a single organization and will communicate with other clients in that group. They will send messages to one another. It is assumed that these messages are more expesensive the more they propagate between servers.

2 Mathematical Basis

This section establishes the properties and mathematical representations used to analyze the problem of this network's efficiency.

2.1 Servers

As mentioned previously, the system will operate upon a network (or cluster) of interconnected servers. The set of servers $[s_0, s_1, ...s_n]$ is denoted as S, with the *nth* servers' contents represented as s_n.

So s_n is the set of all clients that occupy the *nth* server.

In reality, a server will have a limit on the number of client connections it can maintain. We will refer to this as capacity.
Capacity is a function from servers on to $\mathbb{N}$ [cap: $S \to \mathbb{N}$] such that the capacity of server n will be denoted as $cap(n)$.

Then, $|s_n|$ is the number of clients in server n, and $|s_n| \leq cap(n)$

2.2 Clients

Clients are to be organized into 'groups', wherein they will communicate with other clients. For our naive basis, we assume a client will indicate its group to when it first connects to a server, via its request. The server will either process or pass on this request, which will effect how it treats said client.

We denote G as the set of all groups. G_n is then the set of all groups g on server n.

Therefore $|G|$ is the total number of groups that exist, and $|G_n|$ is the number of groups on server n.

A client c is a member of some unique group g, the function $group(c)$ [$group$: $S \to G$] gives us a g such that $c \in g$.

Then G_n is equivalent to $\{\ g : \exists c\ [\ c \in s_n \wedge group(c) = g\ ]\ \}$, or the set of all groups who have members contained in server n.

3 Fragmentation Principle

This is one of our methods to analyze the efficiency of client-server allocation.

3.1 Intro to Fragmentation

We know that server to server communication is expensive relative to communication within a server. By this logic, all the clients in a group should be confined to a single server whenever possible. Taken further, the minimization of group splitting instances between servers is ideal.

The Fragmentation Principle operates on this logic, that fewer instances of group members present on different servers is preferable, as it would minimize the instances of clients communicating across different servers.

3.2 Fragmentation Basis

We want to minimize the instances of group fragmentation, where members of a group are present on different servers, requireing cross-server communication.

G_n and G_m are the sets of groups on servers n and m, respectively.
The fragmentation between them can be denoted as $|G_n \cap G_m|$, or the number of groups present on both server n and m.

The total fragmention of a server n may be calculated as the sum of its fragmentation with every other server:

$fragmentation(n) = \sum_{s_j \in S; j \neq n} |G_n \cap G_j|$

The total fragmentation, across all the servers in S is then the sum of each server's fragmenation.

Total Fragmenation $= \sum_{s_n \in S} \; (\sum_{s_j \in S; j \neq n} \; |G_n \cap G_j|)$
or
Total Fragmenation $= \sum_{s_i \in S} \; (fragmentation(n))$

This interpretation of fragmentation assumes:

- The cost of trans-server communication is unaffected by the explicit distribution of clients. So 1 client on Server A talking to 2 clients on B is no different than 2 clients on Server A talking to 2 clients on B.
- That each case of a split group contributes to the cost. On this assumption, two different groups on two diffent servers absolutely costs more than only one group on the two servers.
- That double counting each split is inconsquential. If members of group A are on two different servers, then they both have a fragmenation of 1, resulting in a total fragmentation of 2. We should be able to divide the total fragmentation by two to resolve this.

3.3 Fragmentation Hypothesis

We propose the fragmenation quantity, detailed above, accurately represents the instances of server-to-server communication via client messages, which only occurs between clients in the same group.

With the given model of fragmentation, we propose that the optimal configuration for the cluster of servers (S) is such that the instances of server-to-server communcation (assumed to be expensive) are minimal.

min(Total Fragmenation) $= min \; [\; \sum_{s_n \in S} \; (\sum_{s_j \in S; j \neq n} \; |G_n \cap G_j| \;) \;]$

3.4 Best Case

The best case scenario is one wherein every group has members located on only one server, such that a server never needs to communicate with any other server.

Best case: Total Fragmentation = 0

3.5 Worst Case

The worst case scenario is one wherein every group has members located on every server, such that a server must communicate with every other server for each of its respective groups.

Worst case(given p groups and n servers):
Total Fragmentation $= (g * n) * n = g * n^2$

4 Implementations of minimization

Now that we have a way to model our goal, we can draft algorithims to optimize the distribution of clients on the servers. In order to analyze the effieciency of these methods, we need to consider both the comparisons and client 'moves' accordingly, as transferring clients will require server to server communication, at the minimum.

4.1 Naively Greedy

For this approach, we will ignore the limitation of server capacity and work greedily to expunge all group fragmentation from each server, one server at a time.

We compare two servers (A and B) at a time, counting the number of clients in a group between the two. This greedy algorithm deals with four basic situations:

- If server A has more members in a particular group than server B, we move the clients from B to A.
- If server B has more members in a particular group than server A, we move the clients from A to B.
- If they have the same number of clients from a particular group, we just move the ones from A to B.
- If either server doesn't have any members from a group, no action is required, as no fragmentaion is present.

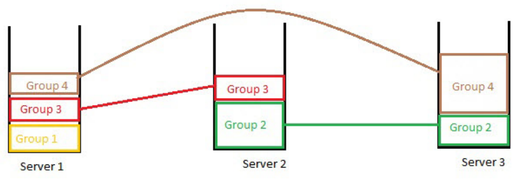
We repeat this operation for every server B that isn't A to eliminate the all fragmentation involving A. To complete this greedy method, we would do that total process for every server, except the last one (we can assume it is fragmentation free if every other server is by the definition thereof)

This requires $(n-1)*|G|$ comparisions per server, if n and $|G|$ are the total number of servers and groups, respectively. That would be $(n-1)^2*|G|$ comparisons for the complete implementation.

We will construct and average of client re-allocations this method evokes as we test it with out simulator, and this data will serve as a reference point for more improved algorithims.

4.2 Just Greedy

The logic of this approach corresponds to the other, but attempts to account for server capacity. This process of server comparison and group allocation is almost identical otherwise. /newline
Now when we coalesce a group's clients on two servers, we will need to verify that one or the other can accomodate that load. We follow the same priority scheme as above, and if neither server can accomodate we try to find the server the the least amount of load necessary to accept the collected clients. The re-allocation only occurs after this verification. If no server can hold them all,



l_ij(f) = number of groups that have members in both i and j (2. in paper)

Here, the sum (sigma) of l_ij = 3

Our objective is to minimize the summation (sigma) of l_ij

Figure 1: This is a visual represenation of the bin packing model.

we divide the clients in half and repeat the search for space to accomodate them.

We have not yet abstracted this in terms of pseudo code, so there are many caveats to account for. Here are some obvious ones:

- This assumes that the cost of fragmentation is not affected by the ratio of grouped clients between two servers. A distribution of 1 client to a server and 9 to another effectively identical to a 5 and 5.
- Load is only considered as a limit to what a server can hold, and not as a possible influence on system efficiency, as fragmenation.

5 Graphic Representations

This section depicts and details models to represent the processes of concern.

5.1 Bin packing model

See Figure 1.

One of the mathematical models we discussed involved looking at:

- Servers as bins
- Groups as packets

These packets need to be efficiently "packed" into these bins As far as possible, we would try to minimize splitting the packet between 2 servers

We would instead, try to re-arrange packets (groups) such that communication between any 2 servers is minimal What we need for this implementation: A constant value that needs to be assigned for each client that is part of a group For example, Client X would take up 25 units of load on a server Then, Client Y would also take up 25 units of load on any given server

Figure 2: This frog was already here and we agreed as a group to let it stay.

6 Leftover notes

6.1 Hypothetical Functions

A function f assigns some members of a group to a server [f: $\cup_{i \in G}\ g \rightarrow S$].

Its inverse, f^{-i} is then a relation of a server to a set of grouped clients.

7 Sources