Greed Protocol -- Greed/1.0

Status of this Memo
    This memo provides information about the UGE Greed
    Protocol. However,it does not specify any details of
    the implementations or technologies used.

Summary
    The UGEGreed Protocol is an application-level protocol
    intended to create a distributed computing system
    built on top of the TCP protocol. The main idea is to
    provide an effective tool for researchers who want to
    test conjectures on a very large number of cases by
    distributing their computations over several machines.

Table of contents

1. Purpose

    The purpose of this protocol is to distribute a large
    computation across multiple servers that are linked to
    each other. Each computation application can be linked
    to another and then become part of the distributed
    computation system.

2. Overview

    The Greed Protocol is based on the TCP protocol.
    Therefore, a Greed connection is in fact a TCP
    connection inside the distributed system and between

the client and the application receiver. In that case, when a packet is sent, the protocol ensures that it will never be lost.

Every strings in the packet of this protocol must be encoded in UTF8 before sending, including the result of a conjecture and the specified address of the client.

In this memo, the notion of parent and children will be mentioned. From an application's perspective, a parent is another application to which it decides to connect. On the other hand, a child is another application that has requested to connect with it.

Every packet in the Greed protocol begins with an OpCode, which is represented by a single byte. An OpCode gives information about the type of packet that was sent. The meaning of each OpCode is detailed in the annex of this document.

A Packet shared inside of the distributed system must contains the address of the client and its conjecture id to identify the source. Please note that each conjecture's identity must be provided by the client as an integer and this identity must be unique across all the conjectures that this client sends.

The Greed protocol ensure that the client will always get all of the requested conjectures unless the client decides to terminate the connection himself or if the target application responds with an op code error, indicating that the application didn't attempt to compute the conjecture.

This protocol also ensure that the system will remain connected even if an application disconnects. The details of the instructions will be mentioned later in this memo.

Since the performance of the distributed system mostly depends on the number of linked servers, a limit has to be set for each application running in the system, about the number of computation that a server can treat, to avoid overloading of the system.

3. Server

   3.1 Setup

   A server can be launch in two differents modes:

- Root : the root of the system. There is only once root per
  distributed system.

- Listening: A server linked to another, it becomes a part of
  the
  computing system and extends the networks.

  Any server can be linked to another and any server
  can accept and distribute conjecture from the
  client.

  By specifying one port for the current listening
  port of the server, the application is launched in
  Root mode. Therefore, if the application is
  launched giving two ports, then it becomes a
  listening application. When a server in ROOT mode
  is shut down, all the servers that are connected to
  the networks will also be shut down.

  If an application receives two different conjecture
  from the same IP address but the conjecture ID
  exists (i.e a conjecture with the specific id for
  the specific IP address is computing), the
  connection will be ignored and closed.

3.2 Distribution

  The server will distribute the interval of
  computation he receives, dividing by the number of
  his children + himself and his parent if he is not
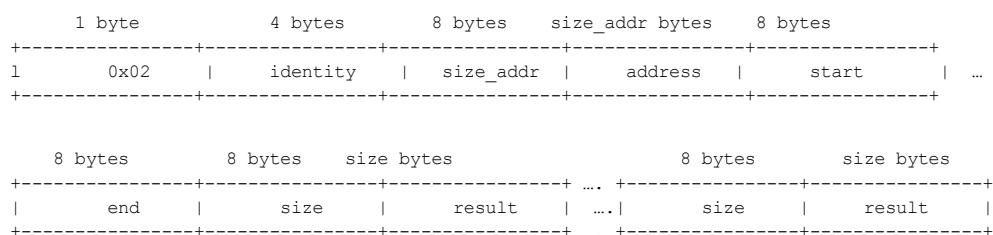 the root application.

  The packet will contain:
- code 1 as a byte
- identity of the calculation as a int
- size of the address as a long
- address of the client
- size of the url of the jar as a long
- url of the jar encoded in UTF-8
- size of the name of the class as a long
- name of the class encoded in UTF-8
- start of the interval as a long
- end of the interval as a long

```
      1 byte          4 bytes        8 bytes    size_addr bytes  8 bytes
   +---------------+---------------+---------------+---------------+---------------+
   |     0x01      |   identity    |   size_addr   |    address    |   size_url    |…
   +---------------+---------------+---------------+---------------+---------------+

 size_url bytes     8 bytes        size bytes      8 bytes         8 bytes
   +---------------+---------------+---------------+---------------+---------------+
…  |     url       |     size      |     class     |     start     |     end       |
   +---------------+---------------+---------------+---------------+---------------+
```

The server will answer to the application that sent
him the packet to
compute with a packet that contains the answer of
the interval:
- code 2 as a byte
- identity of the calculation as a int
- size of the address of the initial client to identify him
  as a long
- address of the client
- start of the interval calculated as a long
- end of the interval calculated as a long
- for each end - start strings the size as a long and then
  the string
  encoded in UTF-8

```
      1 byte           4 bytes         8 bytes    size_addr bytes   8 bytes
+----------------+----------------+----------------+----------------+----------------+
1      0x02     |    identity    |   size_addr    |    address     |     start      |  …
+----------------+----------------+----------------+----------------+----------------+


      8 bytes          8 bytes     size bytes                  8 bytes       size bytes
+----------------+----------------+----------------+ …. +----------------+----------------+
|      end       |      size      |     result     | ….|      size      |     result     |
+----------------+----------------+----------------+ …. +----------------+----------------+
```

When an application gets an answer he will bring up
the answers to the server that receives the request
from the client and this server will send it to the
client.

When a server is overworked he will just distribute
the computation to his childrens and will not take
any part of the interval, if the server has no
childrens he will send it back to his parent until
a server is free to do the computation, a server is
overworked when he gets more than 2 000 000 000
values.

3.3 Response to the client

Once an application that received a conjecture
received results from the other application, he has
to send a packet containing those results and the
range of the computation to the client.

The packet of the response will contains:
- code 3 as a byte
- identity as a int
- start of the interval as a long
- end of the interval as a long
- then end
- start values that is a size of the string as a
long and the string encoded in UTF-8

The description of the packet is in 5.1.1.

3.4 Clean disconnection case

When an application receive a disconnected command,
he will perform a series of operations to ensure
consistent and stable state of the system.

First, the application will update its state from
"ACCEPT" mode to "REFUSE" mode to prevent the packets
requesting computation or connection and will respond to
these request with an error packet containing
only the 0x06 byte. For each conjecture, the application
must send a packet containing the interval of the
response that has already computed to each source of the
distribution. Secondly, this application must send the
address and listening port of its parent to each child,
and each child must then perform a link request using the
LINK packet to connect to the parent.

When a server receives the distribution of a
computation but he is shutdown, he sends a packet
error to the server that asks for the computation,
and this server will compute himself.

```
       1 byte            4 bytes         8 bytes    size_addr bytes    8 bytes
+----------------+----------------+----------------+----------------+----------------+
l      0x07      |    identity    |    size_addr   |    address     |     start      | …
+----------------+----------------+----------------+----------------+----------------+
```

```
       8 bytes          8 bytes         size bytes             8 bytes        size bytes
   +----------------+--------------+--------------+ …. +--------------+----------------+
… |      end       |     size     |    result    | ….|     size     |     result     |
   +----------------+--------------+--------------+ …. +--------------+----------------+
```

When a server that has to send the response of a
computation to a client is disconnected he will
also send the packet of the new address to connect
to the client, and the client will send the missing
interval with the packet in the 5.1.2 section.

The packet to send the address of the parent server
for the client and servers:
- code 4 as a byte
- size of the address as a long
- address encoded in UTF-8
- port as a int

```
       1 byte          8 bytes         size bytes        4 bytes
   +----------------+--------------+----------------+----------------+
   |      0x04      |     size     |    address     |     port       |
   +----------------+--------------+----------------+----------------+
```

When a ROOT server is disconnected he will send a
packet to all the servers of the network, so no
servers will take new calculations from a client,
the servers will end their calculation and then
will be shut down.

The packet to send when the ROOT is shut down only
contains the 0x05 byte.

```
     1 byte
 +-------------+
 |    0x05     |
 +-------------+
```

When a server can't take a new calculation from
clients but receives one he will send a packet with
an error code so the client knows he has to send
his calculation to another server.

```
     1 byte
 +------------+
 |    0x06    |
 +------------+
```

4. Client

First of all, the client must to package their function
to be computed by the server's application into a JAR file
that contains a single class implemented by
a functional interface which takes an int as parameter
and then returns a String. Then this Jar file should be
downloadable from the internet by providing an URL. The
client must specify the URL of the JAR file, the class which
contains the conjecture and the interval of the conjecture,
as 2 long values. It's important to note that if the second
range is larger than the first one, it becomes the final
range of the conjecture. If the two ranges are equal, the
computation starts from 0 and continues up to the given
range. To connect to a server that is part of a distributed
computing system, the client must give the application's
address and port information. Additionally, the client can
specify the name of the final generated file that will
contain the results of the conjecture.
These informations must be specified as parameters of the
program.

Then he has to send a packet containing information
about the conjecture to compute :
-   code 0 as a byte
-   identity as a int

- size of the url of the jar as a long
- url of the jar encoded in UTF-8
- size of the name of the class as a long
- name of the class encoded in UTF-8
- start of the interval as a long
- end of the interval as a long

The content of the packet is detailed in the 5.1.2
section.

After sending the conjecture to the distributed system, the
client must wait for the responses from the connected
application. These responses will be in form of the
'Conjecture response packet' as described in the detail
at the 5.1.1 section. The packet will contain
information about the range of numbers
the treated, as well as the number computation results for
each line of the range. This protocole ensure that the
computation of a conjecture will be completed regardless of
any internal issues of the system. Therefore, the client can
wait until all the requested ranges have been received
before generating a file containing all the results. The
file can be named as specified as argument of the
application or by default 'result.txt'.

5. Annex

5.1 Packet shared

5.1.1 CONJECTURE packet

```
     1 byte          4 bytes         8 bytes          8 bytes        8 bytes      size bytes
+-------------+---------------+---------------+-------------+-------------+------------+
|    0x03     |   identity    |     start     |     end     |     size    |   string   | …
+-------------+---------------+---------------+-------------+-------------+------------+


       8 bytes        size bytes
   +--------------+--------------+
…  |     size     |    string    |
   +--------------+--------------+
```

5.1.2 LINK packet

```
     1 byte          4 bytes        8 bytes    size_url bytes      8 bytes       size bytes
+-------------+-------------+-------------+--------------+-------------+--------------+
|    0x00     |   identity  |   size_url  |     url      |     size    |    class     | …
+-------------+-------------+-------------+--------------+-------------+--------------+


       8 bytes        8 bytes
  +--------------+--------------+
```

```
…  |     start     |     end      |
   +--------------+--------------+
```

5.2. Op codes

There is different opcodes in the protocol:
- 0x00: is the initial packet that the client send to the
  server to initiate a computation
- 0x01 is the packet that servers will use to distribute the
  computation in intervals
- 0x02 is the packet that a server will use to send the
  answer of an interval to another server
- 0x03 is the packet that a server will use to send the
  answer of an interval to a client
- 0x04 is the packet to specify a new address to the client
  when a server is shut down
- 0x05 is the packet that the ROOT server send to all the
  servers when he is shut down
- 0x06 is the packet that a server send to a client or a
  server when he receive a new calculation that he can't take
- 0x07 is the packet that a server send when he is
  disconnected that contains the response that he already
  computed