CS 6210: Project 3

Shubhojit Chattopadhyay(ssc3@gatech.edu) [GTID:902694799]
Abhinav Narain(nabhinav3@mail.gatech.edu) [GTID:902801495]

MultiRPC :

MultiRPC is a parallel remote procedure mechanism implemented in Unix.
MultiRPC is an extension to RPC2 that enables a client to invoke multiple remote servers while retaining the control flow and delivery semantics of RPC .
The motivation for the development of MultiRPC were mostly related to the scalability of RPC.
In order to maintain consistency of files , servers maintain a callback for the files on workstations which notifies when files are changed. An update over a popular file might require a callback of RPC to many workstations which is not scalable.
Moreover, callback to a dead or unreachable workstation must time out before connection is declared broken and the next workstation tried, which will cause a delay of order of seconds which will be unreasonable for widely cached files in case of iteratively making RPC calls. Multicast and Broadcast mechanisms are not scalable because of frequent flushing of caches in clients and servers and also the amount of traffic generated by the broadcast with every remote invocation and callback notification. They might not scale depending on the underlying architecture of the network.
Apart from above two reasons, the use of broadcast and multicast alone does not provide servers with the confirmation that the individual workstation has received callback information, which is implicit in reliable delivery RPC semantics. Hence one requires mechanism that retained strict RPC semantics while overlapping the computation and communication overheads at each of the destinations. Hence the need for the requirement of better remote procedure call mechanism.
Creating normal calls on RPC in different threads doesn't scale well in terms of context swtiching, thread creation, control memory usage etc and so the simple RPC could not be the solution with scalability different(above) issues. RPC2 provides exactly-once semantics in the absence of site
and hard network failures, and at-most-once semantics otherwise
The Coda paper talks about using the multicast in hardware as it suits the Coda File System Architecture(unlike the Andrew File System) with optimization, to make Remote Procedure Calls to different servers.

In this assignment we have modified the xmlrpc protocol which essentially encodes request in xml, to work with different symantics than just one-one RPC call.

Design:

The multirpc design is essentially a server-client model representation of RPC. One client sends requests to multiple servers. Each server then takes time to compute head/tail and responds. The client buffers these results one by one as they are received and then checks the semantic.

Modification of APIs:

The basic premise of the project was to provide the client with a single API which would, under the hood) do MultiRPC. This means, that the client should not realize that this is multi-rpc, but should behave as though it's a single server-client RPC.

The change was very basic. The first one below is the API for Synchronous and the second one is asynchronous.

```
    xmlrpc_client_call_temp( url, methodName,&env, iter_num_of_requests, num_of_servers);


    xmlrpc_client_call_asynch_temp(url, num_of_servers, adder, methodName,
                                   handle, NULL,
                                    "(ii)", (xmlrpc_int32) 5);
```

Each packet sent to the server is now updates a struct:

```
typedef struct st_t {
  int semantic_type;
  int head_count;
  int tail_count;
  int rpc_fail_count;
  int sync_async;
  int avg_time;
} stats_t;
```

Functioning of Multi-RPC:

When the client sends the request to the server, it sends it to multiple servers (based on the script bin/service). The sending is done using a for loop. For synchronous, once a request is sent to 3 servers, all servers respond. This is slightly easier since each request to each server is blocking.

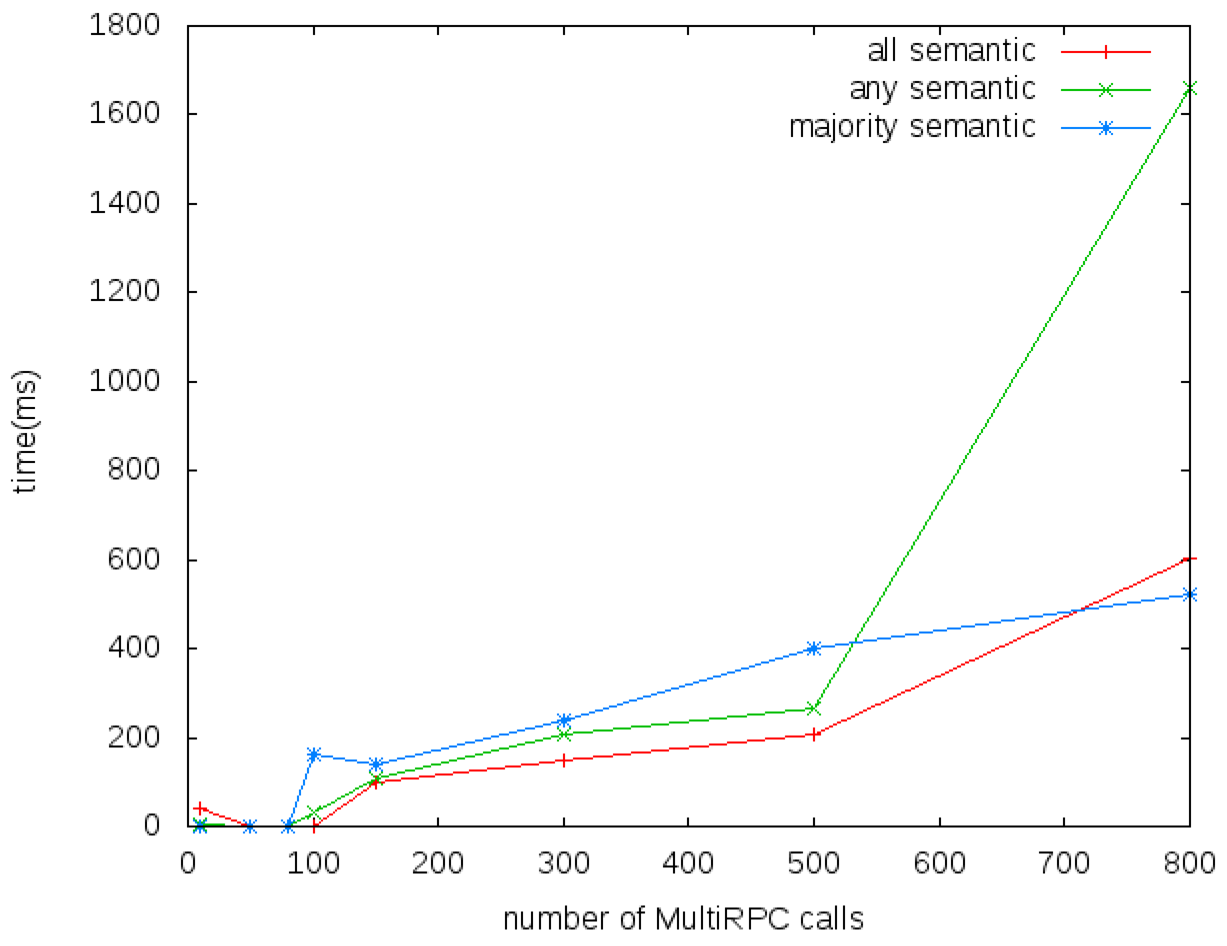
For asynchronous, each request is sent as soon as it arrives at the client. The client then sends it to the servers as soon as possible. It does not wait for responses before sending requests to the server. As soon as the response arrives, the packet is examined for response type and ALL, ANY, MAJORITY semantic.

Optimizations:

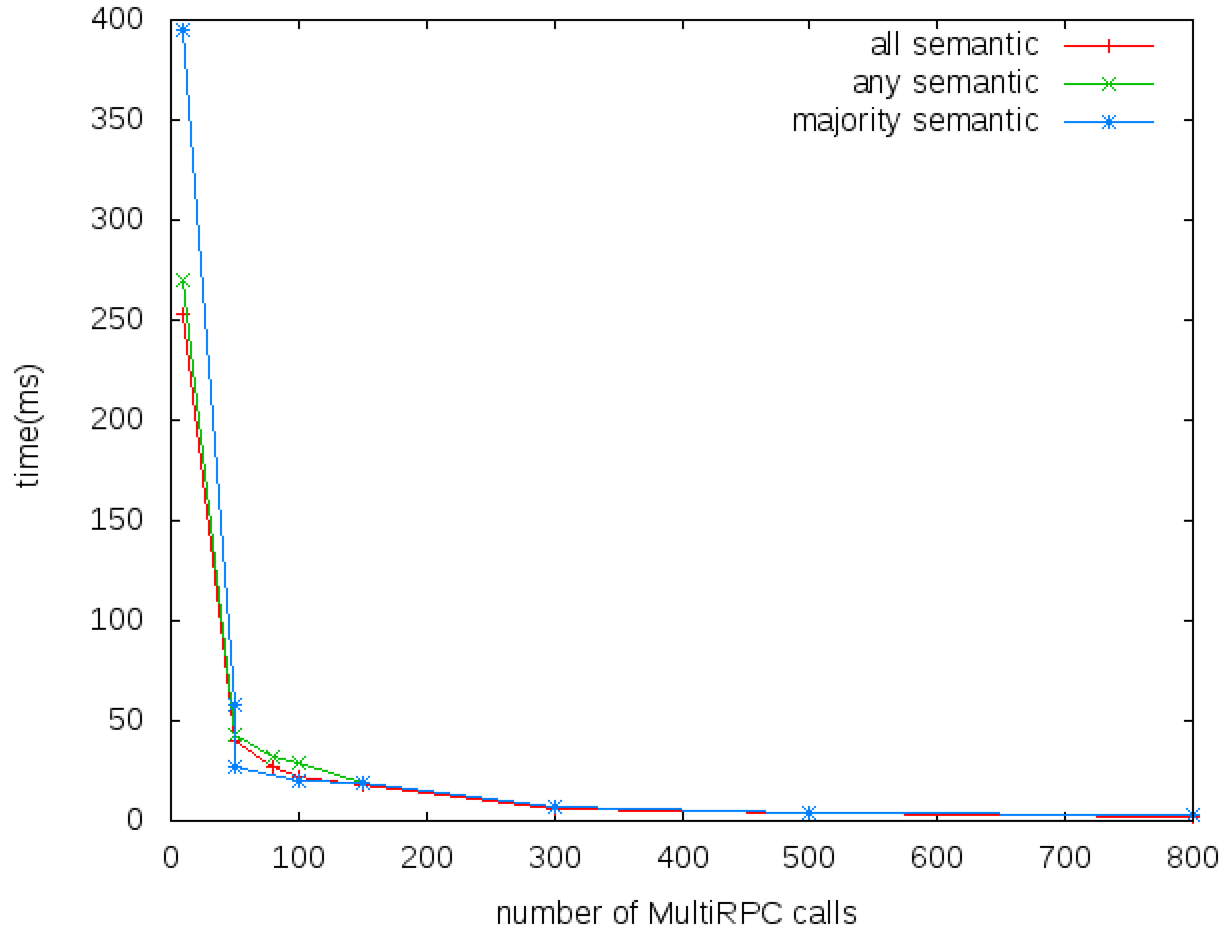
The server is multithreaded to handle multiple instructions coming in at the same time. This multithreading is achieved using inbuilt re-entrant functions in xmlrpc library.

Observations:

Response time for RPC for increasing number of clients :

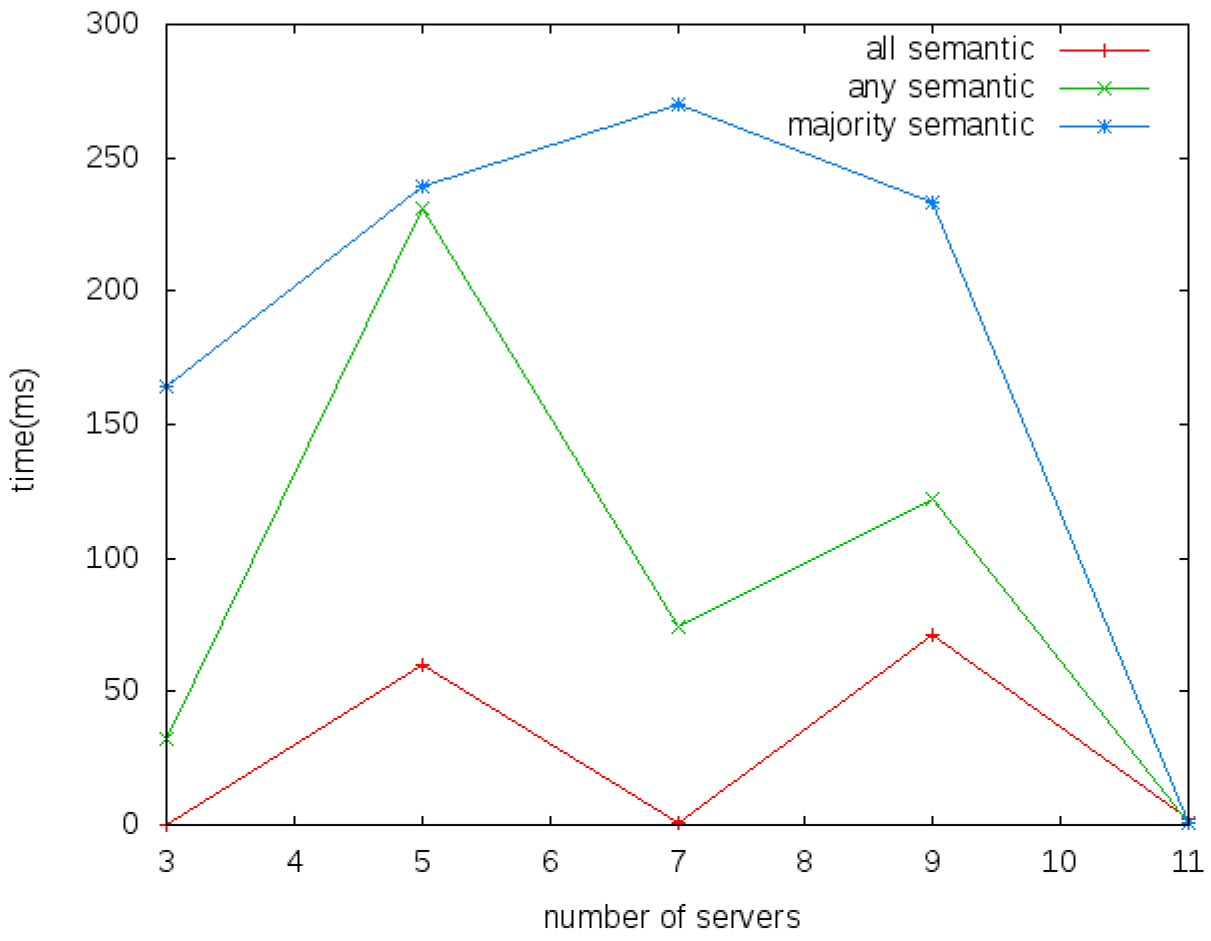


The above graph is for asynchronous RPCS calls for the three different semantics.

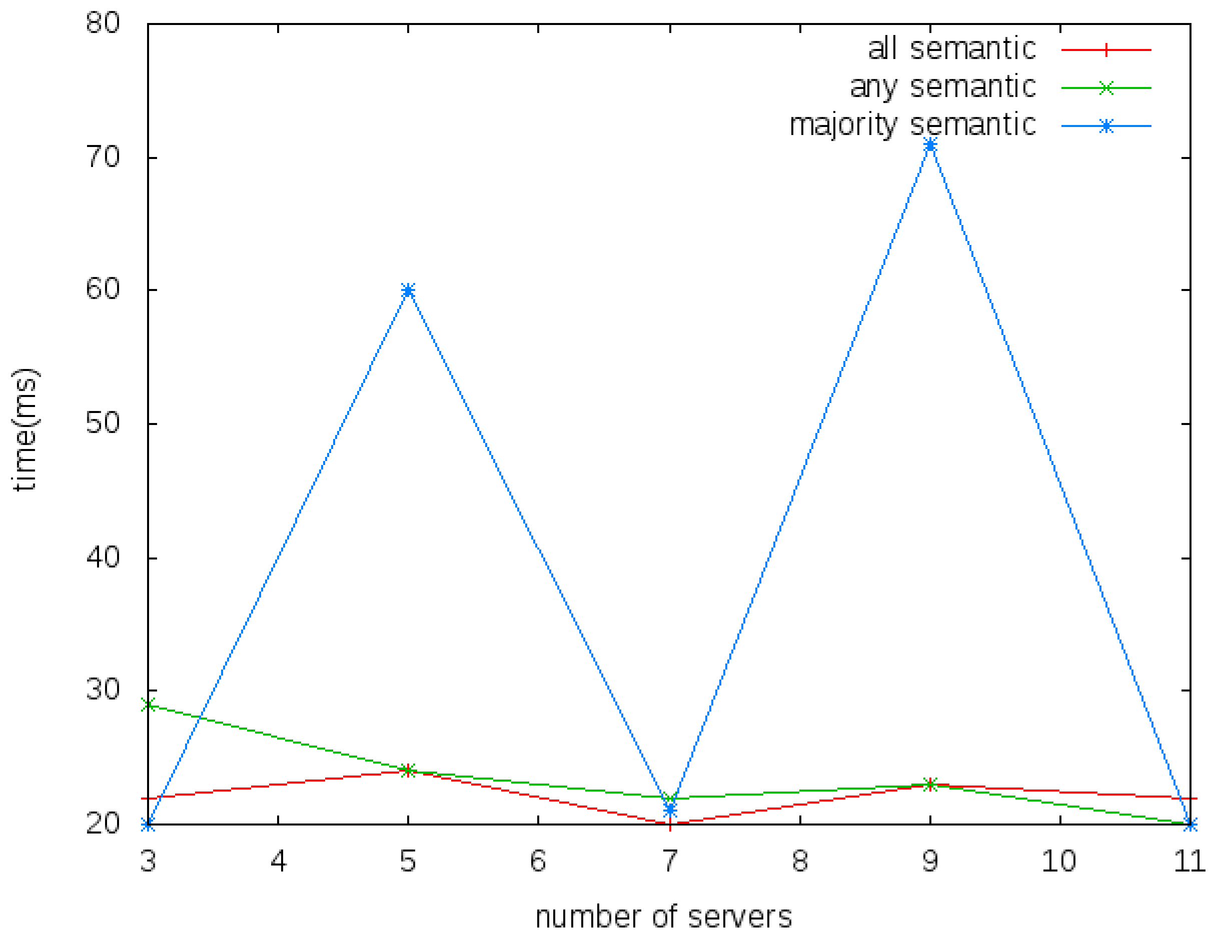


The above graph is for Synchronous RPCS calls for the three different semantics.

Response time for RPC for increasing number of servers :



Response time for Asynchronous RPC calls for three semantics



Response time for Synchronous RPC calls for three semantics

Member Contribution:

Shubhojit:

- Initial Design and planning
- Functional and Architectural design of Multirpc
- Implemented the API for Asynchronous callback
- Implemented the 3 semantics ALL, ANY and MAJORITY
- Designed the packet structure
- Marshalled the correctness of response handling

- Wrote scripts to run and test the design
- Wrote part of the report

Abhinav:

- Initial design and planning
- Implemented Synchronous callback
- Tested the functionality of the design for accuracy and correctness
- Collected statistics and formulated observations
- Wrote part of the report

References:

Coda: A highly Available File System for A Distributef WorkStation Environment
Parallel Communication in a Large Distributed Environment, Mahadev SatyaNarayanan