Generic and Adaptive Load Capture and Replay (GALORE)

INTRODUCTION

change is unavoidable. Like many things, software needs change. But for the most customers, the change in software is still a pain. Some of the reasons that the customers try to avoid change in the software

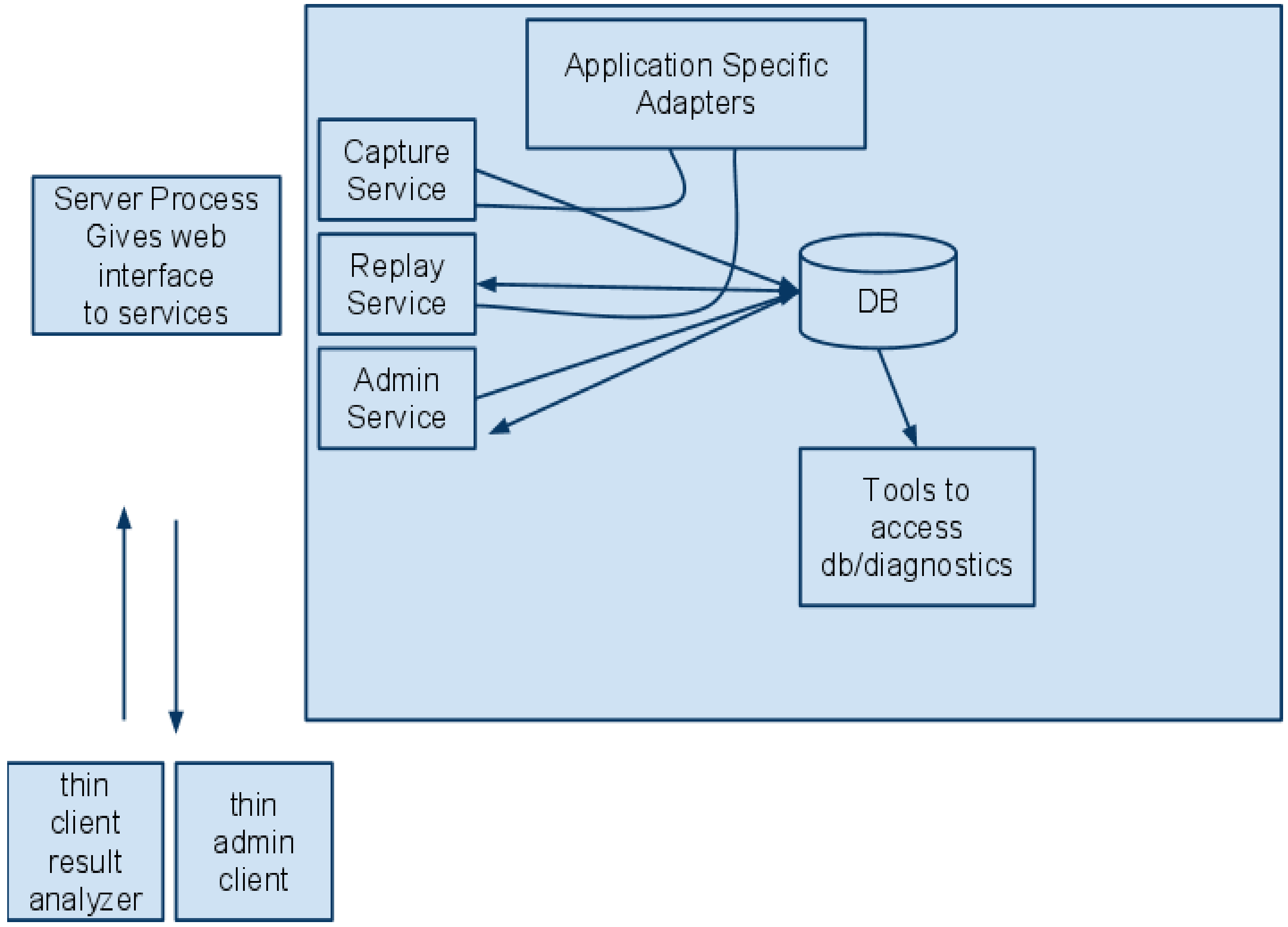
- new software might change the existing functionality in a way that is not intended
- new software might not be stable for production
- at times the new software could be another product in the same category. like replacing a database from oracle to sybase, or replacing a web server etc., In these cases, handling a change would be more difficult
- biggest fear is that their production and test environments are different. the software can not be tested thoroughly to the production configuration.
- Requires more resources to test the changed software, also it might take long time for configuring the clients
- In case of clustering environment, the software upgrade could pose even bigger problem, as more servers need to be upgraded at the same time.
- some of the serious problems may occur only in production systems, but not on test environment.

SOLUTION

solution being proposed would not solve all the above problems, but it solves some of the above problems with limitations(at least in 1.0/or POC).

- The solution works only with client server architecture; meaning, all the clients connect to the server(s) and request for a service and get responses.
- only works for the customers having test and production environments differently
- If the change involves huge client server protocol changes, the solution may not be applicable
- the solution would not do a whole lot of application specific activities while replay.
- One of the main requirements is that the test server should be maintained in the same state as of the production server (before capturing) before load replay begins. This requirement is mainly for the servers with db access.

Here is a rough idea as to how we can develop a solution for this problem. The main idea is to have a generic and adaptive engine that captures the load data and replays in test environment. Following are some components that can be realised while solving the problem.


Application Specific Adapters
Capture Service
Replay Service
Admin Service
DB
Tools to access db/diagnostics
Server Process Gives web interface to services
thin client result analyzer
thin admin client

Load Capturing:

All the servers traditionally listen to a port, the packet capturing service will sniff the incoming packets from the port and start writing the incoming and out going packets to file/database. the component is smart enough to figure out the request/response pairs and writes to the file/db accordingly. the main goal of capturing the packets is to make sure that the overhead is minimum on the cpu and hard disk.
Following is the necessary(there will be more) information to be stored in the database

- source and destination IP address, port
- raw data
- timestamp for request/response

If the data format is open or specified by the application before the capturing(ex: XML, json etc.,), we can allow some kind of filtering before capturing; meaning this will allow only some

particular packets to be captured as opposed to capturing all.
There will be a configurable option ‘dry run’. In this mode, the data will not be stored but will output the diagnostic information like, how much disk space is would be needed if it were to be captured and possibly more information based on that.
the capture can be done in two modes, cpu intensive or I/O intensive, in the former mode, we try to compress the data before writing it to disk.
The capture can also be bound by the disk size, user should be able to specify to capture only load up to 10GB.

The load capturing process can be adaptive, meaning:
we can sample the load capture process with sample data, there by setting up various data definition formats. This way, the capture process can weed out unwanted data and tries not to store those records to disk. This will be of huge help in case of large data transfers and usually helpful for response packets than for requests. It is always recommended to capture all the requests in their entirety otherwise, we risk not being able to replay correctly.
The load capture can be continuous, meaning the load capturing is always enabled by default in the production. It is used only when there is a problem in the production.
How to capture?
dont know what is the best way yet, it also depends on which programming language is being used.
http://eecs.wsu.edu/~sshaikot/docs/lbpcap/libpcap-tutorial.pdf
http://en.wikipedia.org/wiki/Packet_analyzer
Not sure if java could be a better option for capturing, specially in terms of performance. However, it makes sense to do little research on this
http://netresearch.ics.uci.edu/kfujii/Jpcap/doc/

Another issue for capturing would be the security. I dont think currently there is an easy solution to capture network packets without being a root user. Running in root user may not be appealing to larger audience. this should be addressed immediately. we can ask the administrator before capturing to feed the system with a passcode and then we can encrypt the data with 128bit key. This way we can always make sure that the data stored is always safe.

We can also think of following optimizations while capturing data

- we can store the host name only once for the whole capture. In case of multiple host names, we can have a bit map array only once and not store the hostname for every network packet.
- Some of the requests or responses can be hashed and stored only once, this can be made applicable only to small request sizes.(more suitable to HTTP req/responses).
- strip all the unwanted data out of the network packet. Possibly use hash arrays and store small size items only once.
-

Data Storage

It is not decided to use any open source database(couchdb/hadoop/mongo db/cdb etc.,) or write one on our own. What we want is a database

- with no transaction log
- no versioning
- plain read/writes
- scalable
- support data partition???
- read/write remotely
- No SQL

As the writing overhead should be minimal, the writing should be sequential at any point of time. we can not afford to be fancy while writing.
if we were to write our own the db should be hierarchical, should have a header file indicating, an entry for each of the client sockets along with the file where the data for the client is located. Then we can have file sizes of fixed width say(10MB) for the client sockets data. the storage format should be platform independent. Also the database should be scalable and the initial target database size should be 1TB.
The capturing service will be exposed in the thin client, with simple controls like start/pause/stop/sample data etc.,

Load Replay

This is the most complex piece of the whole software. This component is mainly responsible for reading the packet load data from the file/db, and replaying the load.
After capture
Replay is more dependent on the db, as it should know the data layout in the db to optimally read from the db for the replay.
whatever db we may use, the replay should do the following

- first pass
    - If the data has not been summarized, then should go over all the files to create a summarized data(also if possible and not done while capturing, should separate the request and responses data);
    - should go over the summary data, to figure out the number of clients and maximum number of parallel clients(this would tell us how many max threads should be used while replay)at any point of time. Also, it should maintain a list of timestamps as to when new socket/thread has to be started.
- second pass
    - this is where the real action happens, the main thread goes over the list(timestamps and client processes) and start creating a thread based on the timestamp, once a thread is created, it will execute its own commands by reading the data related to it. Again, it will also start issuing the requests based on the

        timestamp(otherwise it will be sleeping)
    - All the threads receive responses from the server and writes the responses back to the file/database, which will eventually be shown in result analyzer.

There will be many more as we keep thinking on it..this is a rough idea :-)
There will be another component for result check(could also be part of replay), to decide if the response is accurate or not. If not it stores the original request&response(in production) and also the test environment req/res pairs.
Also there can be many modes in which user may want to run the replay

- performance testing; then the whole focus is on the response time.
- functional testing; focus on the correctness of the data response.

Again, one of the main building block of the solution is most of the components are adaptive; meaning once the user specifies his/her requirements the replay will be smart enough to use the information in future.
While replaying, we can think of more features like

- partial replays, if the user find a difference, he/she might want to replay only that particular request again.
- can choose two arbitrary times and replay everything happened in between.
- Real time replay; user can start replay while capturing; useful for replication.
- pause/restart; this would be difficult for stateful servers
- while replaying, the user may select to replay only particular sessions as opposed to all of them.
- The replay process should be able to run a shell/command script before the replay starts. The users can invoke any script that they would like to run before the replay begins. Things that can go here are, get the database to the same point when the capture started etc.,
- Replaying the whole workload on only one machine may not be feasible. So, the replay clients can run from different machines. However, we have to partition the data and store them separately while capturing itself.

Following special cases should be dealt as part of the replay process

- Authentication
    - simple authentication, one way hash can easily be replayed; password in clear or one way hash should not be a problem while replaying(no serious production systems have this kind of authentication)
    - SRP/https/SSL; very popular now a days as the client do not have to send the password over the network. http://en.wikipedia.org/wiki/Secure_Remote_Password_protocol#Real_world_implementations These protocols are mainly stateful and they are based on the fact that server and client generates their own private/random generator; So, replaying the whole network data would fail in these cases.
- Session IDs/cookies
    - In a way, session ids and cookies are a mechanism for stateless HTTP servers to be stateful. The session information is stored on the client machines and sent

along with the subsequent http requests.

A way to handle both the above cases is

- to create the connections manually while replaying the network packets. Instead of replaying the packets as is, we can take the information needed like hostname, protocol and other necessary information and create the connection likewise.
- Maintain the session information, while creating a session on replay. Store the session keys/ session ids ; stuff normally stored in cookies. Use the session ids while generating the next request to the server as opposed to use the one in the network packet.
- Give a clean interface for the user to add/change/modify these variables. One good thing about the cookies is that they are all name value pairs.
- With this way, we can support SSL as well.

Application Specific Adapters

As the load capturing and replay modules are generic, there is only little that the replay can do as it has no knowledge about the application. A way to resolve this issues is by letting the users create their own application specific adapters, which will be called while (optional) production load capturing or (most likely case) while load replay on the test machines.
creating application specific adapters can be done in two ways

- through user interface, where the user can just select a packet data element and using some LOVs or simple filtering conditions should be able to create an adapter(this can only happen for replays)
- programmatic interface by exposing a framework for more advanced users. users can write adapters for capturing and replaying. this is a very powerful feature as the users can have the ability to change the requests while replay according to newer software requirements.
- there can be million more reasons why we should have this. Eventually this is where the whole focus would be.

The adapters can be invoked at capture or replay or matching time. May be a single adapter handling all these cases would also be possible.
we can use the latest technologies like osgi, to dynamically install/uninstall adapters.
http://en.wikipedia.org/wiki/OSGi

Result Analyzer

This is main component of the UI design. the client would probably be a thin client(unless there is a road block)
the user should be able to choose a particular run(a load can be run many times), then the

results will be displayed on the screen.

- How is the data in the request/response is shown?
  - if it is xml/json data, we can show the xml format or name value pairs of the data
  - if it is the binary data, then we only show the strings in the binary data, we suppress to show the binary data. However user should be able to configure to show the binary data as integer etc., in which case user will enter the position in the data segment and also the type, so that we can display the same accordingly.
  - This is the most important aspect of the product, the easier we show it to the user, more usable would be the software.
- the responses that differ from the production will be shown, highlighting the place in the data segment where it is different. Immediately next to it, should have an option to ignore the difference. If the user chose ignore, then all the places where we have a similar difference should be smartly ignored(or may be ignore all could be another option)
- user can select a data packet and would like to do ignore similar packets. this is very important for user as he/she does not have to go thru lot of such packet data. this has to be done smartly as we should figure out the data format of the specified packet and do a smart comparison with all the other error packets to ignore them.
- users can group some packets and give a meaningful name, In which case we should show only the summary of that group and the user can dig into it, in case required.

Data Analyzer

As we know what kind of protocol is being used (IMAP/HTTP/SNMP/SMTP/POP3/SOAP/LDAP/DNS), the software can start making predictions about the data format, (atleast for known protocols, that should not be a problem). Also, the user can create his/her custom protocol and save it. The software can use it adaptively based on that and avoid lot of noise while matching the results. In addition to that, we can make intelligent guesses about the data, which will help the users to write the application specific drivers easier. Also, the fact that there are lot of known and widely used protocols makes a strong case for the product.

GOALS

- - cpu 5% overhead target
- - I/O 5% overhead
- -70% of the problem should be dealt by generic engine
- -Minimum app specific code
- -Simple and easy framework to write app specific drivers
- -Interoperability between two similar apps
- -Response time change analysis
- -Analysis/Predictions w.r.to different scenarios like I/O throughput, more cpu cores etc.,

TRENDS

currently, more and more software is being hosted as a web services. web services for the most part use open standards for communications. As the data format will be known ahead of the time the generic/adaptive load replay can be more useful.

TERMINOLOGY

should come up with nice and meaningful names for the product and different components in the product(even for the product galore sounds like crap :-)

RESEARCH

study large web server/application server/complex client server software to get an idea about the communication style and data formats being used for communication.

CURRENT ALTERNATIVES

the current solutions are not generic and mainly db specific.
oracle db replay
ireplay : http://www.exact-solutions.com/products/ireplay

BUSINESS USE CASES

- helpful for cases, where we can not automate the tests(for variety of reasons). It would be simple to install GALORE software and start testing while GALORE is capturing the request, responses from the server. Once the capturing is done, it can be replayed many times, highly useful for regression testing. This could be a very powerful tool for QA/developers. For developers, then can make a change and before committing, easily replay this tool.
- One obvious use case, as its been mentioned many times in the document, bringing a software change in the production environment. Users simply install the software on the production and test systems. While in production, the software captures the data, which can be replayed later on test systems. The results of the replay can be analyzed in nice GUI, with lots of additional features/options.
- Some of the serious problems can not be reproduced in test environment for variety of reasons. GALORE should help (if not all the problems) to reproduce some of the problems in test environment. This is possible due to the fact that GALORE will replay

the production load in similar way as it happens in production.

- Useful in replicating the load between master, slave servers. GALORE can capture load on the master server and starts replaying it on the slave in the real time. While replaying all errors will be logged.
- Interoperability between two servers??? If it is required to change from one web server to another in production
- Highly useful for cases where there is hardware change or OS change, OS upgrades, OS patches in production.
- Can also be used for client testing, the same test case, with roles reversed. we replay the server for the client.
- load characterization. we can extract patterns from the captured data and come up with interesting statistics like,
    - %of errors of type (say if it is http page not found or server internal errors etcs); we can categorize the errors
    - find out the performance bottlenecks for some application patterns
- As we can have adapters for replay, many times the user can modify the adapter instead of making changes to the application and immediately test this(by running the replay) to see if there would be any performance improvements or bottlenecks with the changes. This is a super useful usecase for some customers who do not want to make changes to the application without knowing ahead of the time if these changes are valid or not. The other approach to do this is to let the user do the capture again with the new app (in test env), now we can extract the patterns from the new capture and use the new captures with the old data to run the replay(this is a hard problem to solve,we should see how much we can do this)
- Another use case could be, after capturing the production data, we can diagnose the load and come up with possible bottlenecks in the application based on some standard guidelines(ex: for http, yslow - http://developer.yahoo.com/performance/rules.html), then we can also make those changes automatically before replay(as a replay adapter) and show the performance gain.
- one other use case could be (as a side effect of result handler module) is ;
    - imagine a scenario; say a customer accessing BOA site, complaining that he is running into some issues while accessing some features of the BOA website. As the customer support person do not have any idea as to what options the user is clicking that resulting into the errors, it would nice if the customer support engineer can see a live replay of what the user is doing(even with a bit of lag), to figure out the possible cause of the problem, this might not all the time result in finding out all the problems, but it can surely detect some issues.
- Using Galore, we can come up with some unique functional test cases, and provide them back to the testing team to add them to their functional test suite. This has many advantages like
    - we can always have exact production test cases to be tested while functional testing the future software upgrades.
    - we can come up with unique functional test cases(not all captured) and add only

them to the functional test suite.

- this minimizes the functional regression testing effort for future software/hardware upgrades.