

dRIZ:

distributed Reed Imitation Zephyr

A design proposal
for extensions to the RIZ server system.

Gavin Weld White

April 8, 2002

1 Why dRIZ?

In a potentially large network of users, the reliability and scalability of the RIZ service becomes an increasing concern. For this reason, some modifications to the RIZ server structure may be necessary. Due to the locational transparency of the RIZ service, there are some restrictions on the ways in which the server structure can be extended. Nevertheless, it seems appropriate and feasible to establish multiple RIZ servers as an ad-hoc network maintaining a distributed hash table of users.

In the first version of RIZ, there is no built-in support for multiple servers, so anyone wanting to RIZ someone else must be logged on to the same server as they are. This means that either the number of people logged on to a server will grow as the number of people using RIZ, until someone decides to set up their own RIZ server. Those wishing to RIZ only their small group may migrate, and find their isolated server to be sufficient, but those users wanting to be in constant contact would log on to all available RIZ servers. Either way, when a single RIZ server goes down all of its state information is lost. While it might be possible to maintain a central lookup table of who-is-logged-on-where, this faces the same problems as the single-server approach. It is clear, therefore, that multiple servers are needed.

2 dRIZ Clouds

My approach to having multiple servers is to use a modified CAN [R00]. Users are hashed to keys that determine their location in the CAN, and servers are responsible for maintaining disjoint subsets of the CAN whose union is the whole CAN. For example, such a CAN could use the i^{th} letter of the username as the key for the i^{th} dimension of the hashtable. While this method would be

inefficient, it will be used to demonstrate the feasibility of this system, given a hashing scheme.

We consider two different types of processes, clients and servers. A client is a process run by a user to access the services of the RIZ system. A server is a process run by an administrator to provide the services of the RIZ system. It is possible for both client and server processes to use the same machine. It is also possible to run them on separate machines.

When we speak of a user doing something that involves a server, we mean that they do this through the client, possibly in a way that is transparent to the user. Similarly, when we say that a server does something with a user, we mean that the server does this with all of this user’s clients.

Each user may have multiple clients, and each client will have only one server. Each server may serve multiple users, and will have a subset, not necessarily proper, of its users’ clients. A user may have multiple servers if the user has clients that are sufficiently distant from each other in the network topology.

Given that a CAN resolves names to locations, and the names used by RIZ are locationally transparent, the distribution of users in the CAN is without reference to network location. As such, we allow multiple CANs, each of which replicates the entire namespace of the users.

A server joining the dRIZ cloud would choose a location in the namespace at random, and ask any other server in the cloud to route it, using “pass” messages, to the server currently providing service to that location. When the new server finds the server responsible for its chosen location, they may engage in a mutual authentication protocol — not to be implemented in this version of DRIZ, but in the next one.

Once the servers are sufficiently authenticated to each other, either the new server is incorporated into the CAN, a different CAN is contacted for the new server to join, or a new CAN is created. If the latency between the new server and the existing server is higher than an established threshold, the new server will join or create a different CAN. Details on this can be found in Section 3. Otherwise, the new server will assume responsibility for half of the namespace of the existing server, as described in [R00] and below.

The server that is splitting its namespace then sends a message of type “pass” to all users in the portion of the namespace that is going to the new server. It also sends a “pass” message to the other neighbors of the new server, indicating the new server as a replacement neighbor, and replaces the appropriate neighbor in its directory with the new server. Finally, it sends a message to the new server with the IP addresses and ports of two new neighbors in each dimension of the CAN.

3 Weather Systems

When a user logs in to any server, they are passed off to the appropriate member of the server cloud using the “pass” message, as described below, to complete the login process. Clients keep track of a list of their server’s neighbors, and in

Type	From	To	Body	Meaning/Action
login	server1	server2	$u \in_{\mathbb{R}}$ username-space	route server1 to location $\mathcal{H}(u)$
pass	server	client	serverID:IP:port	login to new server at IP:port
pass	server2	server1	u :serverID:IP:port	contact serverID for $\mathcal{H}(u)$

Table 1: Message Type Table.

particular which neighbor to go to if the server dies. When a server needs to pass a user to another server, whether to get out of the CAN, to add another server to the CAN, or for login, it sends a message of type “pass” to the user’s clients, and they relogin to their new server. During login, such a message would be sent before mutual authentication, so as to minimize required traffic. Clients logging in to a new server due to a server crash would include such information in their login message, which could then be verified by the new server, while servers trying to log out would be expected to notify their neighbors first, and then their clients.

The servers themselves would also need to keep track of each other to some extent. This functionality could be provided by a new class, serverCAN, that is connected to the userTable. If a server receives a message for a user it is not responsible for, it can throw it in the CAN. The CAN would route messages among servers, and, if appropriate, pass them to its own server for handling. One benefit of using the CAN design is that each server only has to keep track of $2d$ neighbors, where d is the number of dimensions in the CAN, and $\frac{u}{n}$ users, where n is the number of servers in the CAN and u is the number of users in the DRIZ.

All messages, then, would be thrown in the CAN, and the CAN would then pass them to the appropriate server. The appropriate CAN would ask its server to handle the incoming messages for which it is responsible.

One of the concerns raised in relation to having multiple RIZ servers is that of routing relatively local messages through a distant server. Because usernames bear no relation to geographic or network topology, this topological name resolution issue is necessarily pushed to the server level. For example, if a RIZ server is established in Tokyo, Japan, and it is connected to a RIZ server in Portland, OR, there might be some users in Portland who are passed to the server in Tokyo, or vice versa. For these users to have to send local messages through a distant server seems absurd.

Since the username itself gives no locational information, servers routing messages to a user must have some other way of resolving names to locations. If the usernames are distributed among the RIZ servers without reference to location, there always may be a case where a user is placed far from their actual location in the network topology. One solution to this would be to distribute all usernames among each set of local RIZ servers, with any user logged in to a “distant” server having a reference to that server’s CAN as their location, while “local” users would be distributed among local servers as described above. Thus, there would be multiple “local” CANs, connected by references to each other.

In short, it is suggested that we extend the RIZ server functionality by adding a modified CAN structure, and requiring clients to accept a new message type, “pass”, having as its body the IP and port of the next server to try. With suitable choice of timeouts for defining “local” and “distant”, this extended RIZ server system should be infinitely scalable and flexible. Multiple distance thresholds could be used to distinguish between interstellar and planetary networks. Whereas groups of servers working together have often been called “clouds”, a group of servers implementing this doubly-distributed system should be called a “weather system”.

References

- [R00] Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S., “A Scalable Content-Addressable Network”, ACM Tech Report, Berkeley, 2000