Prioritized Scalable Distributed Concurrency Services

Goals

Supporting prioritized mutual exclusion on a distributed system by way of implementing the extensions to known protocol for prioritized mutual exclusion for distributed systems. The extensions are meant for intent locking support coupled with read, write and upgrade lock modes. Enhancing the concurrency protocols for middleware services.

Current

Constant concurrency level experiments are being run for more than 90 nodes in the system on IBM SP. Variable concurrency levels, constant ratio experiments are also being run for more than 90 nodes on IBM SP. Writing for JPDC'03 is in progress. Understanding TAO implementation and reading in progress.

Work Done (During Fall, 2002)

- Simulator was enhanced to add configurability with respect to request mix specification, inter-arrival and critical section lengths, random priorities, switching to prioritized and unprioritized version of the simulator, timing granularities of milliseconds or microseconds.
- Porting the simulator to AIX, LINUX, studying MPI, implementing the protocol with MPI instead of TCP/IP for AIX 4.3.Support for MP_THREAD_MULTIPLE was deemed unnecessary.
- Student research competition and poster submission, acceptance and attendance at OOPSLA 2002, seattle.
- Poster submission and acceptance at JGI, 2002.
- Submission to ICDCS 2003 with the comparison of our protocol with Naimi's protocol. For fair comparison, Naimi's protocol can be seen with dual perspective: pure protocol with single lock, same functionality protocol with N locks. Discussion of the entire protocol for the first time in a publication. Result waited.
- Submission to IPDPS 2003 with the results obtained by experiments on IBM SP with MPI. Results include the constant ratio, variable concurrency effects, comparison with Naimi's protocol and adjusted request latency. Result waited.
- Introduction of constant concurrency level in the experiments to observe the response time behavior and understand it. It can be concluded that the response time behavior is inherently super-linear and can't be better. Some interesting theoretical explanations for the response time behavior.
- Started on the TAO implementation. The model is understood but it is required to understand some of the relevant aspects of the CORBA specification and APIs related to concurrency services.

Work Done (During Last Month)

- The hypothesis about getting logarithmic response times by way of keeping the concurrency level constant was proved inaccurate. The experiments done for constant concurrency levels indicate that even if the concurrency level is constant

(fixed number of requests active in the system at any time), the response time is not better than linear.

- The experimental results also imply that the clear logarithmic message overhead behavior is lost when the system operates at constant concurrency levels. This is due to the fact that the propagation path of the request message is prolonged due to decreasing probability of the intermediate nodes being able to grant the request.
- The message breakdown of the constant concurrency level experiments clearly shows that the overall message overhead behavior is all due to the request propagation. While other types of messages get stable at reasonably small number of nodes, the request message overhead keeps on increasing more like linearly.
- The experiments with constant ratio, variable concurrency levels were conducted from 30 nodes up till 90 nodes on IBM SP. As expected the message overhead behavior even for very large number of nodes is stable and logarithmic. However, the response time behavior is super-linear and more such with lower ratios of waiting time to the inter-request arrival times.
- These results point to a distinct behavior of the protocol: Higher the degree of concurrency better, more stable and logarithmic the message overhead. On the other hand, higher the concurrency, worse and more super-linear the response time. Improving the message overhead behavior worsens the response times and vice versa.
- One possible argument is that, the total response time of each request is composed of two elements: the transit and propagation time of the requests when it is forwarded from one node to another, and the queuing delay when it is queued locally by some node due to incompatibility. Now, the former element can be derived from the message overhead behavior while the later can't be. While the request is queued, there are no messages being passed and so, message overhead doesn't reflect this time that is only seen in the final response time of the request. So, as the concurrency level increases, more and more time is spent in the local queues and less and less messages are passed. This gives us better message overhead behavior while the response time keeps on increasing.
- The queuing delay can be estimated by estimating the queue lengths with varying concurrency levels. The detailed estimation function is derived and matches the super-linear behavior of the response times. This shows that the response time behavior is inherently super-linear O(n**2).
- In the TAO front, after communicating with Bala (Washington University), the framework is clearer. It seems that each client can instantiate its own Concurrency Service object and use the API provided with that. Thus, the protocol can be embedded into those APIs and hidden from the clients. It is still not clear how exactly the concurrency objects will communicate with each other. Probably they can register themselves with the IOR with specific naming conventions and so they can identify each other, get references and invoke some operations on remote objects, which are the handlers. However, in the ORB net, there might be some of the clients not intending to use the concurrency service. Now, if they are accessing the server object concurrently with other clients, which will interfere with the protocol and mutual exclusion cannot be guaranteed. So, clients need to be aware of the server objects that are concurrent.

Future Work

- Report the latest findings to the Journal of Parallel and Distributed Computing special issue for middleware 2003. Write, integrate and submit.

- Experiments with Linux cluster for large number of nodes and comparison with Naimi's protocol for the same.
- Make progress on the TAO front.
- Thesis committee should be formed and formalities taken care of.
- Requirements for this semester regarding thesis preparation depend on the advisor.

Nirmit Desai
nvdesai@unity.ncsu.edu
Date: 12-10-2002