SOL: Skoll on OverLays

Cristian Lumezanu

University of Maryland, College Park

CMSC 838P

Distributed Continuous Quality Assurance

- emerging trends and challenges in software systems:
  - user-specific customization
  - increased cost and reduced development time
  - distributed and evolution-oriented development
  - explosion of configuration space
- in-house quality assurance is no longer enough
- solution: take quality assurance to the next step

Skoll

- general, around-the-world, around-the-clock, feedback-driven process to carry quality assurance tasks
- client/server architecture
- smart generation of QA tasks based on the configuration model
- automatic characterization and visualization of results
- however, there are some disadvantages...

Server Overloading

- the server is the authority in the system
- creates tasks, distributes them, collects and interprets the results
- for each client:
  - runs planning algorithms (which can be computationaly hard)
  - keeps connection state (which can require a lot of storage space)
- computational power and storage are limited

THE SERVER CAN SERVE ONLY SO MANY CLIENTS

Compilation Overhead

- no information is kept about already compiled sources
- when only the runtime configuration options change, the code will be recompiled
- clients have no way of knowing who has the compiled sources in this case
- e.g. ACE+TAO
  - P4, 3GHz, Linux FC2, 1h
  - Celeron, 1.1GHz, Windows XP, 4h
  - the download of the code from CVS can take more than 30 minutes

TIME AND BANDWIDTH ARE WASTED

What Can We Do?

- "when it doesn’t scale, build a tree"
- clients should know about each other
- more, clients should cooperate
- no more dumb, obedient clients; make them smarter

Proposed Solution

- advance one more step
- push part of the server functionality towards the clients
- have the client organize themselves in an overlay network
- follows the approach of peer-to-peer systems...
- ...and, hopefully, inherits their advantages also

Advantages of Peer-to-Peer Systems

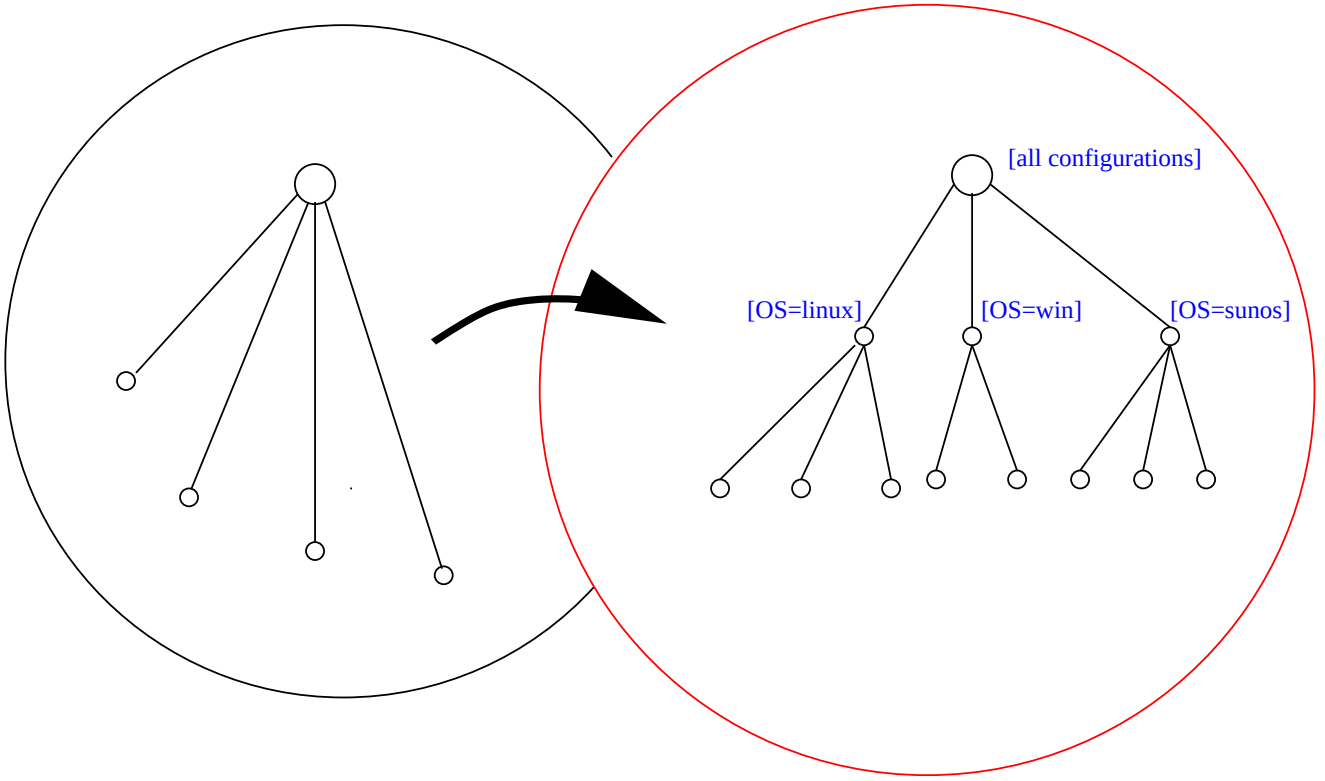
- scalability -> the system will support a larger number of clients
- robustness -> the functionality of the system will not be affected by the failure of one client
- self-organization -> the clients will organize themselves in such a way cooperation between them is enabled and encouraged
- availability -> a client does not always have to resort to the server for any information, it can get it from other clients

ALL THESE POINT TOWARDS A MORE COOPERATIVE DCQA

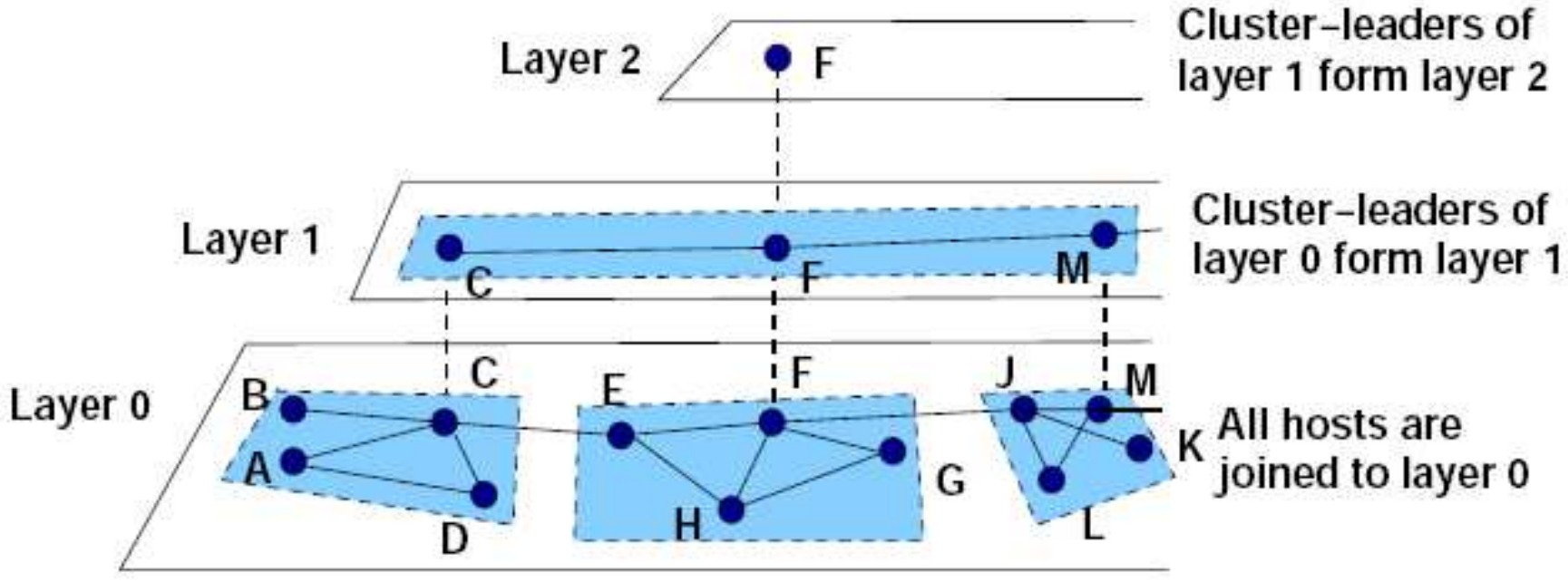
How to Do It?

- migrate intelligence to the periphery of the network
- "clients" have part of the server functionality:
  - other "clients" can connect to them and request jobs
  - they run planning algorithms locally on a subset of the configuration space
  - they run adaptation strategies
  - they send the partial results to other clients or to the server
- the server retains its functionality but part of its work is taken over by some of the clients
- configuration space is divided according to the topology

SOL Overlay Tree

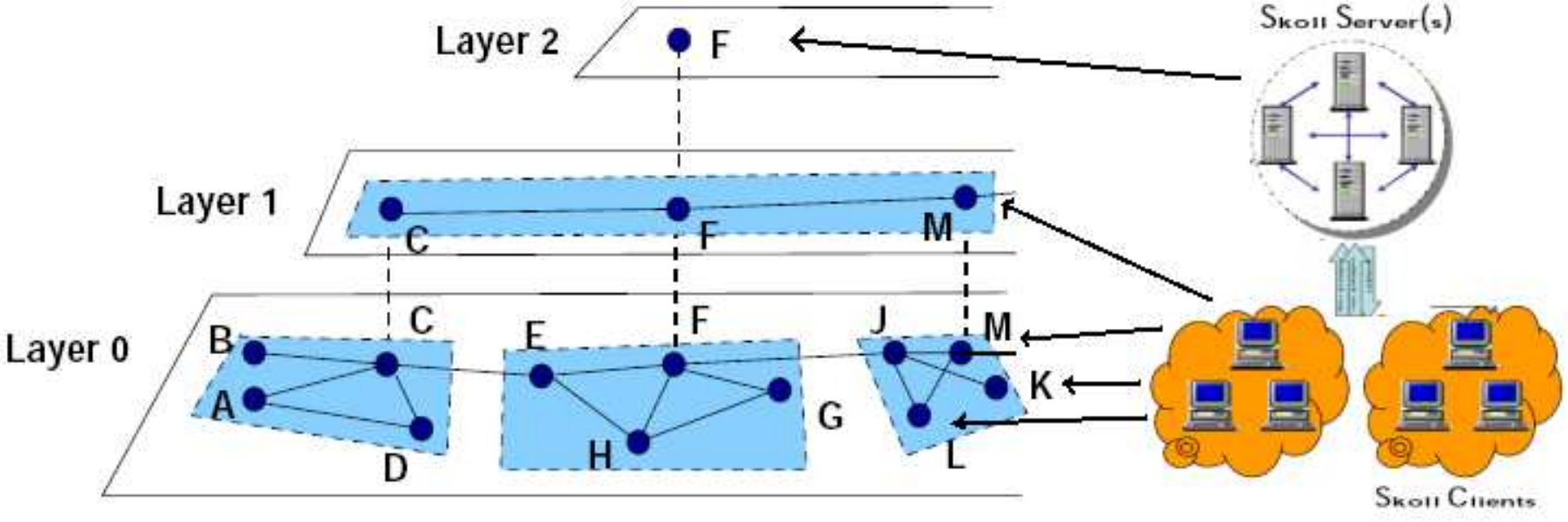


NICE

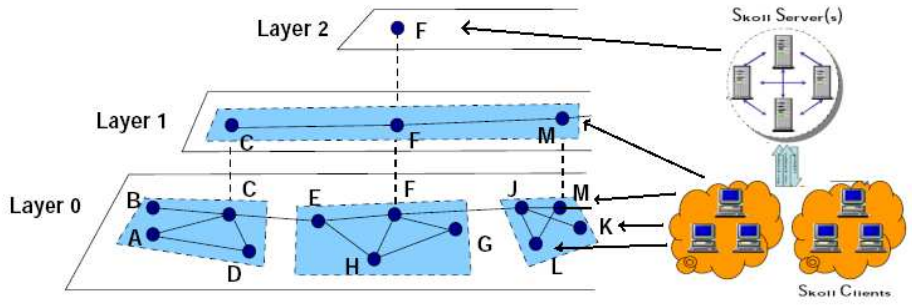


The NICE Project: http://www.cs.umd.edu/projects/nice

Deploying Skoll

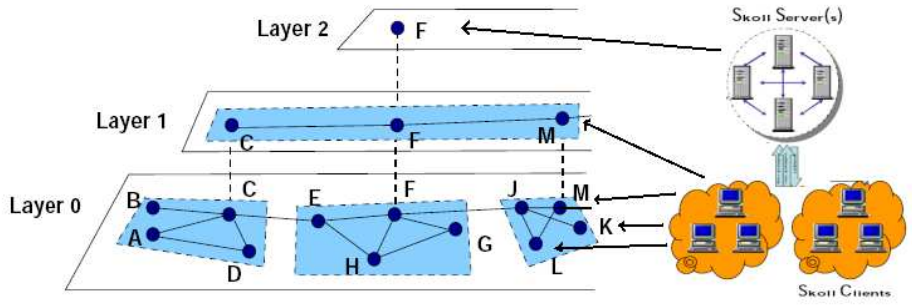


Improving Server Scalability



- each new client connects to the server
- then, it will request a subtask from the "closest" client which is in a higher layer (cluster leader)
- number of connections and state are O(logn) instead of O(n)

Improving Compilation



- a client can request an already compiled code from another client in the same cluster
- thus a piece of code is compiled once and used by everybody with the same compile options
- distributed compilation triggered by cluster leaders can reduce the compilation time even more

Final Thoughts

- the integration of Skoll and NICE is not trivial
- how to assign parts of the configuration space? where exactly should we drawn the line?
- can we have a completely decentralized architecture?
- do we need security guarantees?
- can we provide an incentive for clients?