CCOF AND COMPUP2P

PRESENT BY RAN LIU
CMSC 818K
APR 19TH 2011
CCOF: MOTIVATION

- A variety of users and their applications need additional computational resources
- Many machines throughout the Internet are idle for large periods of time
- Users are willing to donate cycles
CCOF: CLUSTER COMPUTING ON THE FLY

- P2P based cycle sharing system
- Highly dynamic environment
  - Join/leave
- Why not central server?
  - Not scalable
  - performance bottlenecks and traffic hot spots
- Why not DHT?
  - Dynamic resource information cannot be naturally hashed
DYNAMIC HOSTS

- Peers can join and leave at any time
- Peers can withdraw cycles
  - Cannot be used to compute
  - But still can route to other peers
- Avoid unnecessary migration
- Peers may crash/die and disappear
PROFILE BASED MODEL

- The profile is based on observation of user daily routines
  - i.e. idle at night, emails in the morning etc.

- Such usage patterns can be acquired manually or from a monitoring program
WORKPILE APPLICATIONS

- Consume huge amounts of compute time
- Master-slave model
  - Master gives out code to many hosts
  - Each host computes
- Embarrassingly parallel- no communication among hosts
SEARCH ALGORITHMS

- Evaluated five methods
  - Centralized Search: ideal
  - Expanding Ring: breadth-first
  - Random Walk: depth-first
  - Advertisement: capability rather than requests
  - Rendezvous Point
    - Optimal placement
SCHEDULING STRATEGIES

• Hosts have maximal available time at night
• Reschedule at night or give up
Fig. 2. Job completion rate under uniform workload, when the ratio of clients to donors is 0.1.
Fig. 5. Job completion rate under uniform workload, when the ratio of clients to donors is 0.7.
Fig. 7. Job migration failure rate under uniform workload, when the ratio of clients to donors is 0.7.
CONCLUSION

• Rendezvous Point perform best overall
• To address larger-job-issue: avoid favoring large jobs.
• But it may cause larger jobs to starve
• The message overhead is stable for Rendezvous point
COMPUP2P: AN OVERVIEW

- CompuP2P is a peer-to-peer (P2P) utility infrastructure
- Dynamically build markets for a computing resource
  - Market Owners (MO) do the arbitration
- Uses game theoretic ideas to govern pricing of computing resources
- Usage
  - Provide computation capabilities to processing-intensive user applications, like network simulations, graphics
  - Support storage intensive applications such as data-bases and file systems
SYSTEM MODEL

• Assumes a P2P configuration that uses Chord for addressing and peer connectivity

• Nodes are selfish, earn profit by selling their computing resources

• Resource Units
  • Compute power: cycles/sec for $T$ time units
  • Memory storage: giga(mega) bytes for $T$ time units
CONSTRUCTION OF COMPUTE POWER MARKETS

• Markets for different amounts of compute power are created

• A market deals in only one type of commodity.
  • Commodity here refers to compute power in a certain well-defined range

• The same node can be responsible (i.e. be a market owner MO) for running multiple markets

• Two schemes
  • Single overlay
  • Processor overlay
Markets for: C=0, C=1, C=2
Sellers: 1, 5, 2, 6, 3, 4
SINGLE OVERLAY SCHEME

- The number of CPU cycles/sec (C) gives the Chord ID of the market and the successor is the MO
  - \( MO = \text{successor}(C) \)
- Simple to implement
- Can lead to uneven assignment of markets among nodes and requires large number of hops
PROCESSOR OVERLAY SCHEME

• More uniformly assign markets among nodes
  • MO = successor(hash(C))
• MOs form an additional overlay
  • IDs equal to the commodity values
• The lookup returns the IP address of the market trading in commodity equal to or greater in value than requested
Chord overlay
(numbers next to the nodes are the Chord IDs)

C = average idle capacity in cycles/sec

1, MO for C=3
2
3
4, MO for C=1 and 2
5
6

3
4
5
6

CMID=1

Instance of node with Chord ID=4

Instance of node with Chord ID=1

PROCESSOR OVERLAY SCHEME

Processor Overlay
INCENTIVES TO SELLERS

• Use of marginal costs is the optimal pricing strategy
  • No profits

• Fixed list pricing
  • MO always gets the same cut
  • Reverse Vickrey Auction
    • Lowest bid wins at next-to-lowest price
    • Service profit = next-to-lowest bid - lowest bid

• MOs are selfish too
VARIABLE LIST PRICING

\[ \text{Payoff}_{MO} = \frac{(MC'_N - MC'_1)}{(MC'_N)^2} + \delta \]

\[ \text{Payoff}_{seller} = MC'_1 + 1 \]

- MO is paid more for lower cost services to ensure lowest bid wins
- Payoff covers cost of service
- Total cost to buyer is bounded

\( MC'_N, MC'_1 \) Refer to the marginal cost values of the highest and lowest cost supplier
PROTOTYPE IMPLEMENTATION

• Users submit a task-specification XML file as input
  • Describe sub-tasks, inputs etc.
• KARMA protocol to handle currency exchange
  • Each node is given a starting amount
• Fault-tolerance
  • Handling node crashes
    • Dynamic checkpointing
    • Select a storage node store periodic checkpoints
FAILURES HANDLING

• MO Failure
  • sellers relist, clients go elsewhere
  • Sellers periodically relist themselves to a new MO

• Seller Failure
  • MO checks seller, and purges dead ones
  • clients don’t pay for unfinished work

• Client Failure
  • refer to log file, contact computing nodes
  • The computing nodes will discard the results after a certain time period
  • pay?
CONCLUSION

- P2P system for resource arbitration
- Dynamic construct markets
- Uses game theoretic ideas to govern pricing of computing resources
- Based on Chord, scalable, efficiency and robust
- Great for large simulation jobs
THANKS!
REFERENCE
