Towards Real-Time Volunteer Distributed Computing

Sangho Yi[1], Emmanuel Jeannot[2], Derrick Kondo[1], David P. Anderson[3]

[1]INRIA MESCAL, [2]RUNTIME, France

[3]UC Berkeley, USA

Motivation

- Push towards large-scale, soft real-time applications
  - Online games: Chess, Go
  - Real-time multimedia processing
  - Interactive visualization
- Desktop grids
  - Cost-effective
  - **BUT** limited to high-throughput applications

Goal

- Develop and evaluate prototype to support soft-real time applications on Desktop Grids
- Approach
  - Real-time server-side algorithms for task management [previous work]
  - Full implementation and evaluation of algorithms
    - Parameter-based admission control
    - Deadline timers
    - Case study with Go and Chess

Software Context: BOINC

- Middleware for Desktop Grids
  - Led by David P. Anderson at UC Berkeley
- Underlying infrastructure for projects such as:

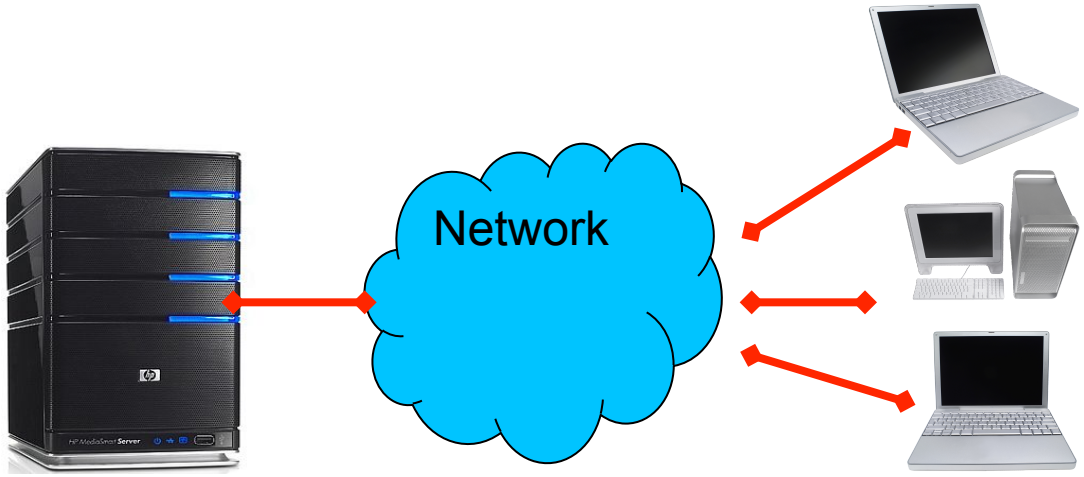
SETI HOME

world community grid

Rosetta@home
Protein Folding, Design, and Docking

BOINC Components

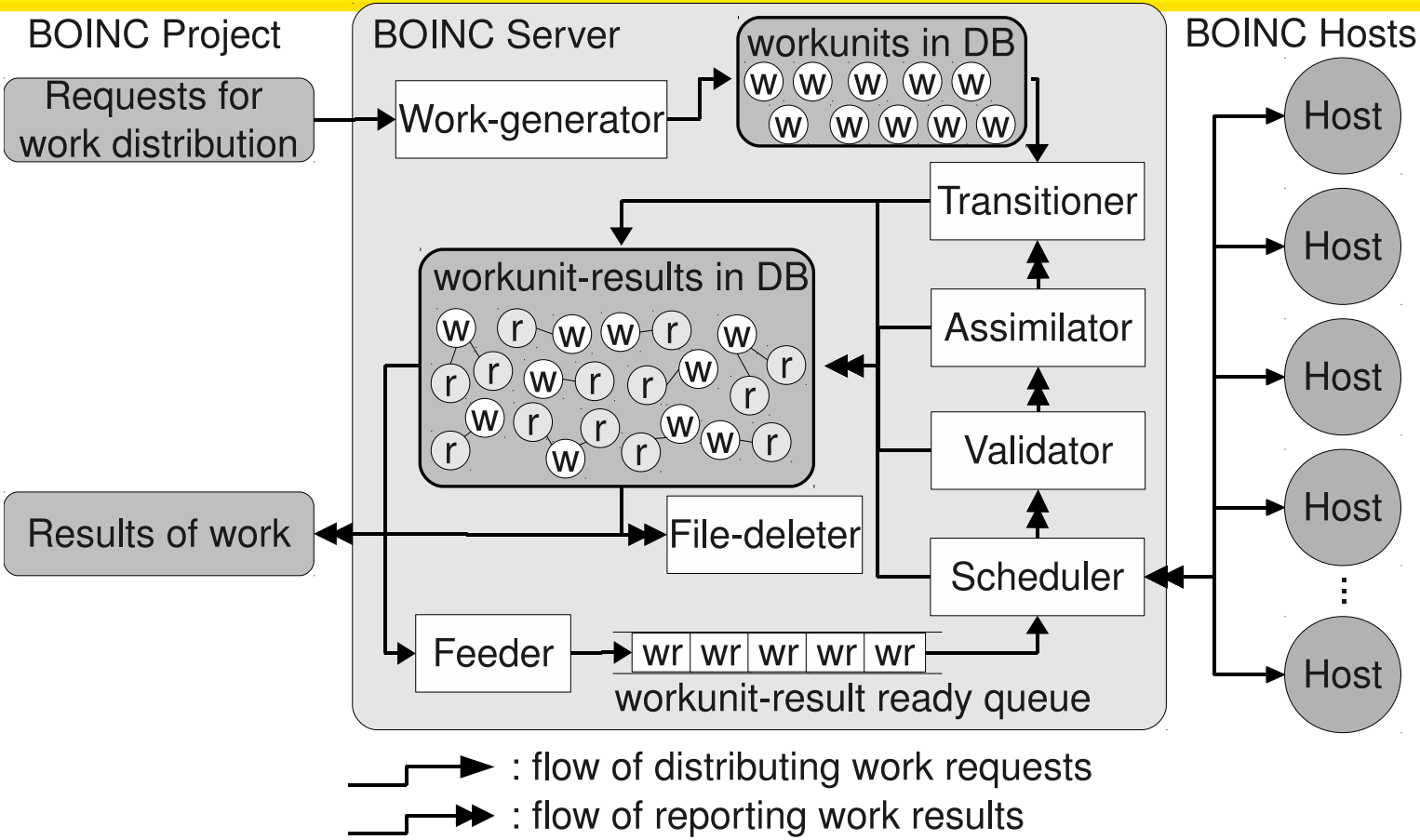
- Server
  - Web server, database, management daemons
- Clients that compute workunits
- Network between server and client



BOINC Limitations

- Maximizes throughput, not minimize latency
  - Performs 1-50 transactions / sec [http://boincstats.com]
  - Sends relatively large computation to clients lasting hours or days
  - Does not have real-time [RT] guarantee
  - Same with XtremWeb, Condor, GridBot

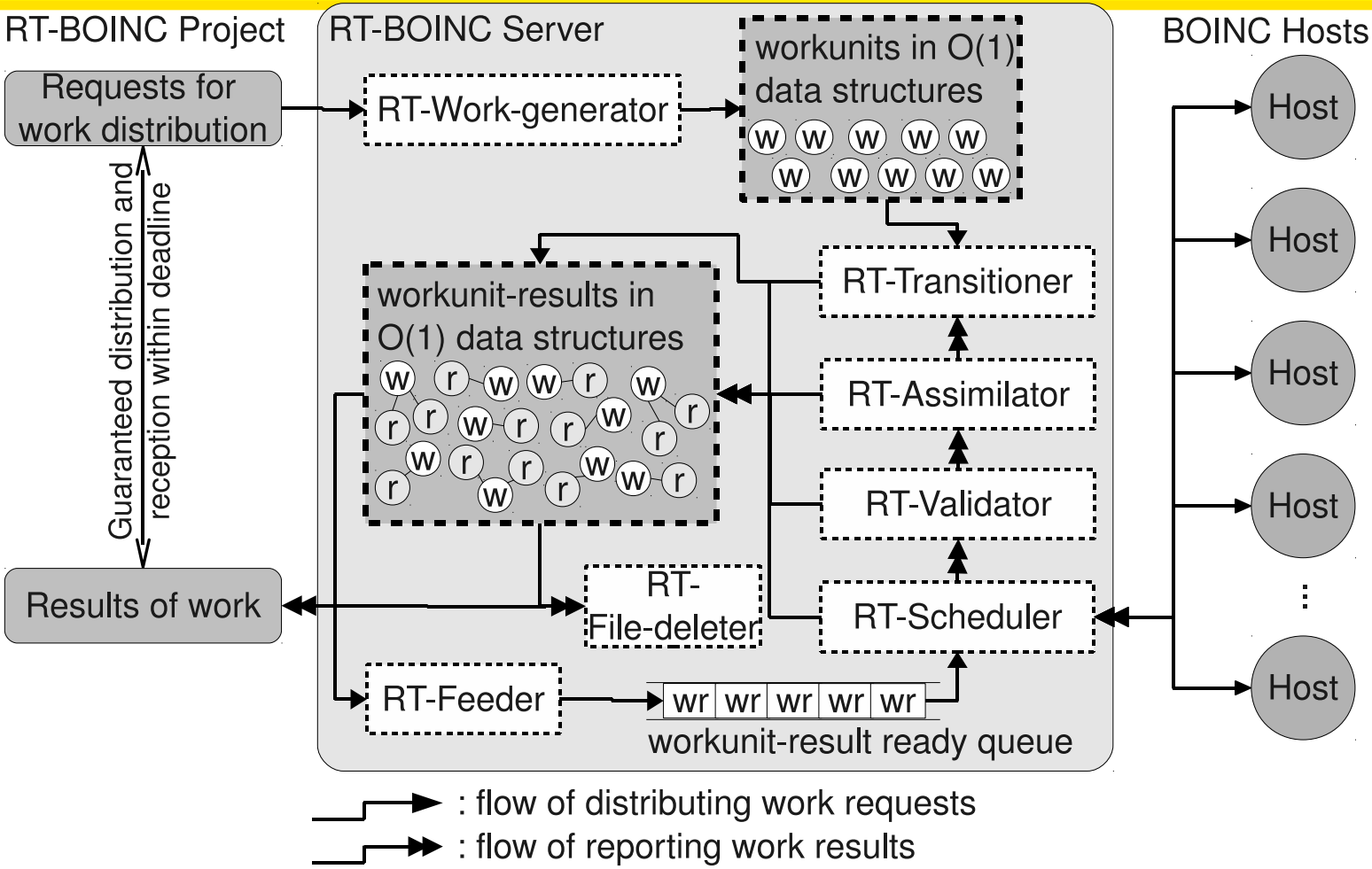
BOINC Internals



RT-BOINC

- Provides low worst-case execution time (WCET) for all components
- No database operations at run-time
- O(1) interface for data structures
- Reduces complexity for server daemons
  - Close to O(1)

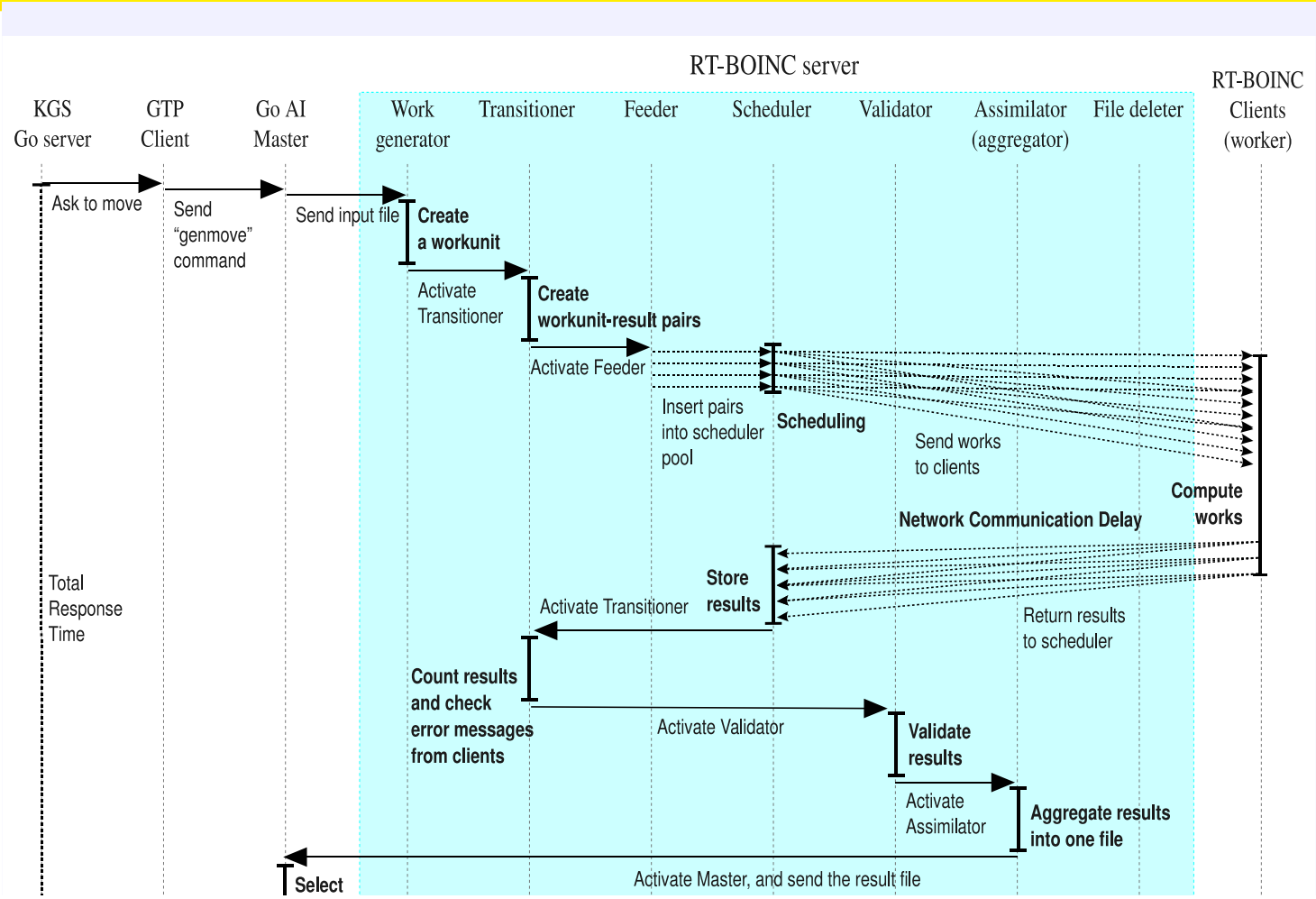
RT-BOINC Internals



Implementation of RT-BOINC

- Used BOINC as the basis
- 99% backward compatible with original BOINC
- Full source code and sample applications available at
  - http://rt-boinc.sourceforge.net
- New key features
  - Admission control
  - Deadline timer

BOINC Workflow



Admission Control

$$T(r, n_c) = w_c(r, n_c) + \sum_{k \in S_d} w_k(r) + w_{tr}(r) + \frac{2 \cdot n_p \cdot w_{sc}(r)}{n_c}$$

Variable	Definition
r	task request
n_c	number of available server cores
w_k	WCET of the kth server process
w_c	worst-case network delay between server and client
S_d	set of all server processes
T	worst-case task completion time
w_{tr}	WCET of transitioner
w_{sc}	WCET of scheduler

Deadline Timer

- Timeout to deal with potential deadline miss due to
  - Unavailability
  - Non-responding clients
- Workunit generator initiates timer
- Timer triggers validator when it expires
  - Validator validates with all currently collected results
    - In-progress results are ignored and discarded

Performance Evaluation

- Case study with real-time applications having both soft and hard time constraints
  - Go
  - **Chess**
- Evaluated on experimental platform Grid'5000

Platform Description

Platform detail	Description
Processor	2.0GHz (8 threads) Intel Xeon E5504 (2 dual-quads)
Main memory	8×4GB (1066MHz) dual-channel DDR3
Secondary storage	1 TB disk (7.2K RPM)
Network interface	1 Giga-bit Ethernet
Operating system	Ubuntu 9.10 (64-bit) kernel version 2.6.31-19
Web and database	Apache and MySQL released in Aug. 2010
RT-BOINC detail	**Description**
Lookup tables	Three-levels tables first, second: 4 bits, third: 8 bits
Number of records	50K for each table in-memory data structure
Volunteer resources	Grid'5000 hosts (64-bit) grenoble, nancy, rennes, and sophia sites (40 $\sim$ 800 cores)

Chess Engines

- Many chess engines are available
  - Some play at near-professional levels
- Most communicate with a GUI or a standard protocol called UCI (Universal Chess Interface)

Stockfish

- Open-source
- Portable (Mac, Linux, Windows)
- Multi-threaded
- Very strong
- Comes with opening book

Chess AI

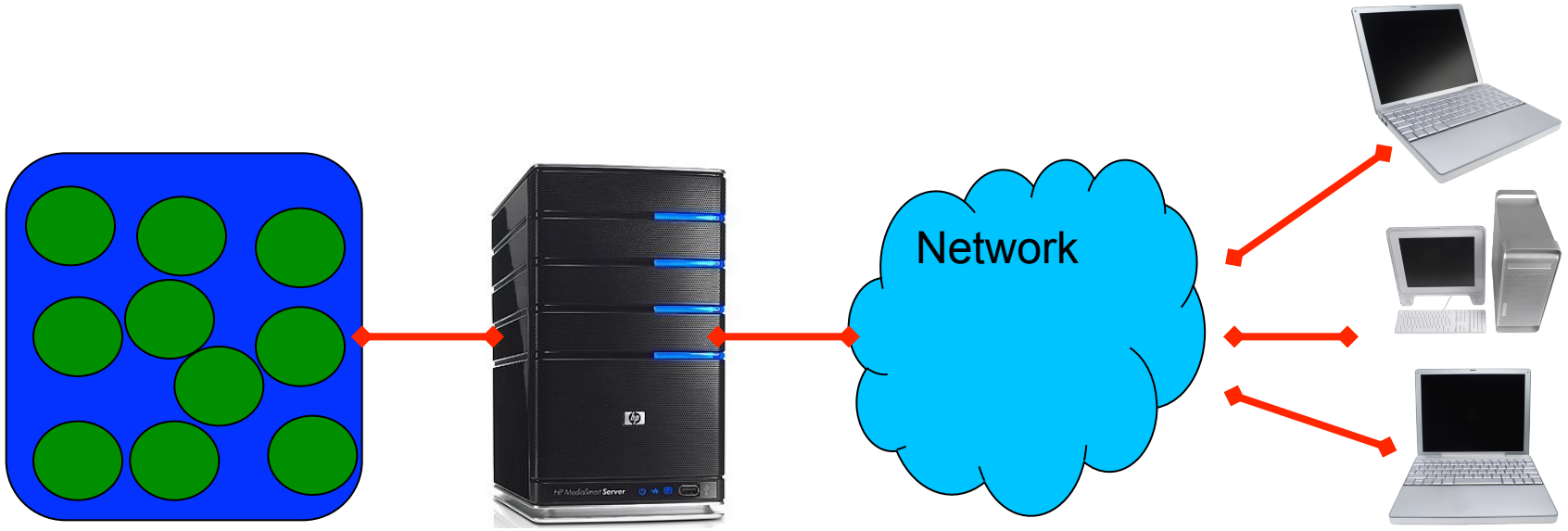
- Tree search
  - Each node is a position
  - Each edge is a move
- Evaluation function of position
- Pruning techniques to accelerate search
  - Min/max, alpha/beta
- Time management
- Opening book

Chess Constraints on Desktop Grid

- No communication among workers
- Minimal communication between workers and server (task and result)
- Potentially many workers
- Churn
- Some workers may never return with answer
- No time to generate a different task for each work unit
- **Impossible to use min/max algorithm in this context**

Chess on Desktop Grids

- Server has a position
- RT-BOINC should compute the best possible move in a given amount of time
- How to distribute search of tree among workers?

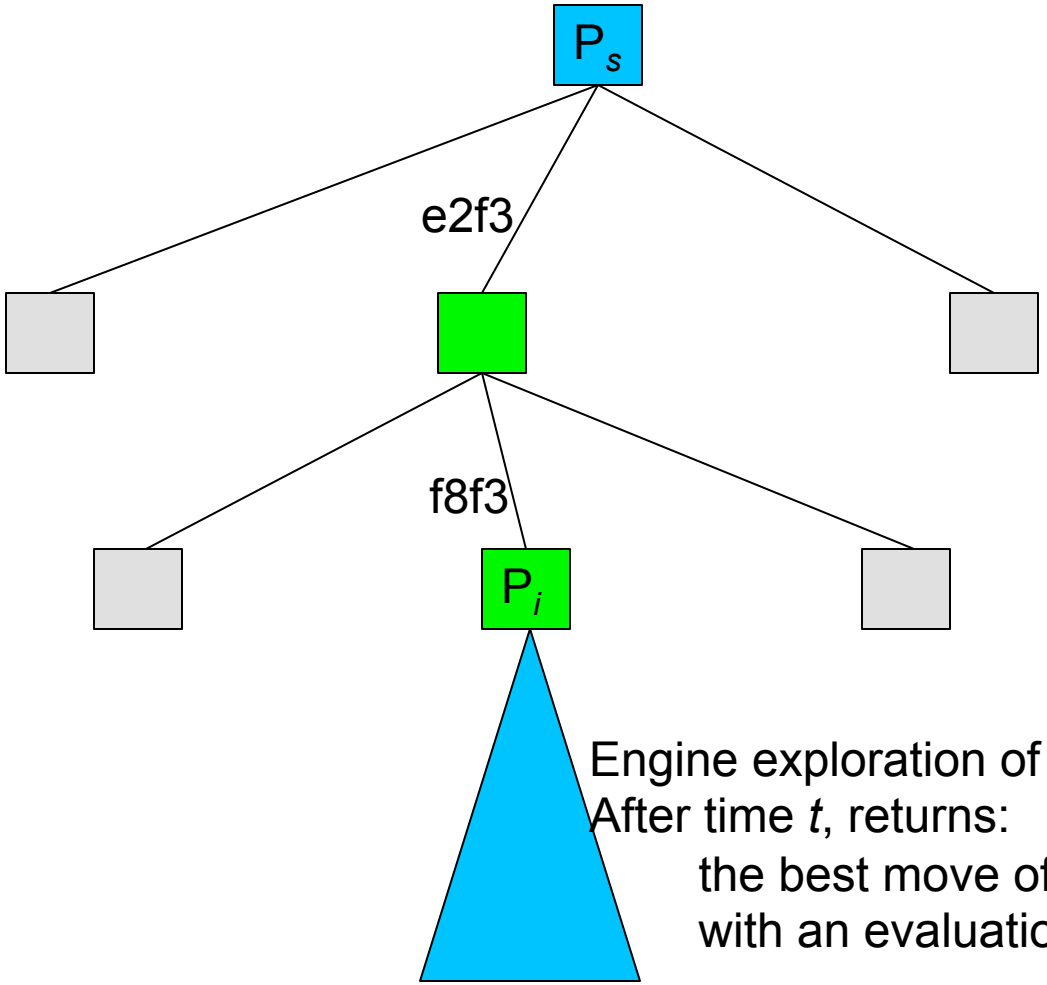


Solution: a randomized algorithm

Principle:

- Each worker receives:
  - the same position P_s,
  - an integer *n* and,
  - a soft real constraint ***t*.**
- Each worker *i* plays *n* moves at *random*: reach position P_i
- Each worker *i* asks the chess engine to process P_i until time *t*
- At time *t*, the worker returns the best move found by the chess engine with an evaluation and the *n* moves required to reach P_i
- The server aggregates clients results using min/max algorithm and compute the best result for position P_s with its evaluation

Example (n=2)



The workers returns:

- P_s
- e2f3 ; f8f3
- P_i
- g2f3
- -4.5

Engine exploration of the subtree from P_i
After time *t*, returns:
the best move of P_i (e.g. g2f3)
with an evaluation (e.g. -4.5)

Server aggregation

Get all workers answer

Aggregates answers to built a tree *T'*

Apply min/max on *T'* to determine the best move

Question: how far is T' from the real tree T?

Uniform Random Choice

How many workers are required such that almost surely all the nodes of depth n will be considered?

Hits: the average arity of a chess tree is 30.

At depth n we have $m=30^n$ nodes. If we assume that each node is selected uniformly.

Formally, I have m coins in a bag, I pick one uniformly and replace it. How many picks do I need such that there is probability ε that I did not pick all of them at least once.

Uniform Random Choice (Cnt.)

Approximation of the solution:

1/m: probability that a node (a position) is explored

1-1/m: probability that a node is not explored

$(1-1/m)^p$: probability that a node is not explored after p trials

$1-(1-1/m)^p$: probability that a node is explored after p trials

$(1-(1-1/m)^p)^m$: probability that all nodes are explored after p trials

Example: n=3, m=27000, ε=0.01

$(1-(1-1/27000)^p)^{27000}=0.99$, $p\approx400000$

n=2, m=300, ε=0.01

$(1-(1-1/300)^p)^{300}=0.99$, $p\approx4450$

Towards a Bias Random Choice

Uniform random choice:

- Requires many workers to be sure that we did not miss a good move

Not all choices are equivalent

Need to bias the search towards “the best” moves.

Question: how to rapidly estimate what are the best choices?

Best Moves Fast Estimation

Evaluating a move at a fixed shallow depth (5), is fast (orders of ms).

How good is this evaluation?

In general it appears to be good but this is not always the case.

Evaluating the Quality of Depth 5 Evaluation

Question:

- Given a position
- The best move for this position
- What was the rank of this move at depth 5, among all possible moves?

Experiment:

- We took 1407 positions from the (all games of 1972 Spassky-Fischer world championship match, removed opening book positions)
- Most of them are tight positions (21 games, 11 draws)
- The engine analyzes each position for 20 minutes
- We record the rank of the best move when the search reaches depth 5

Results

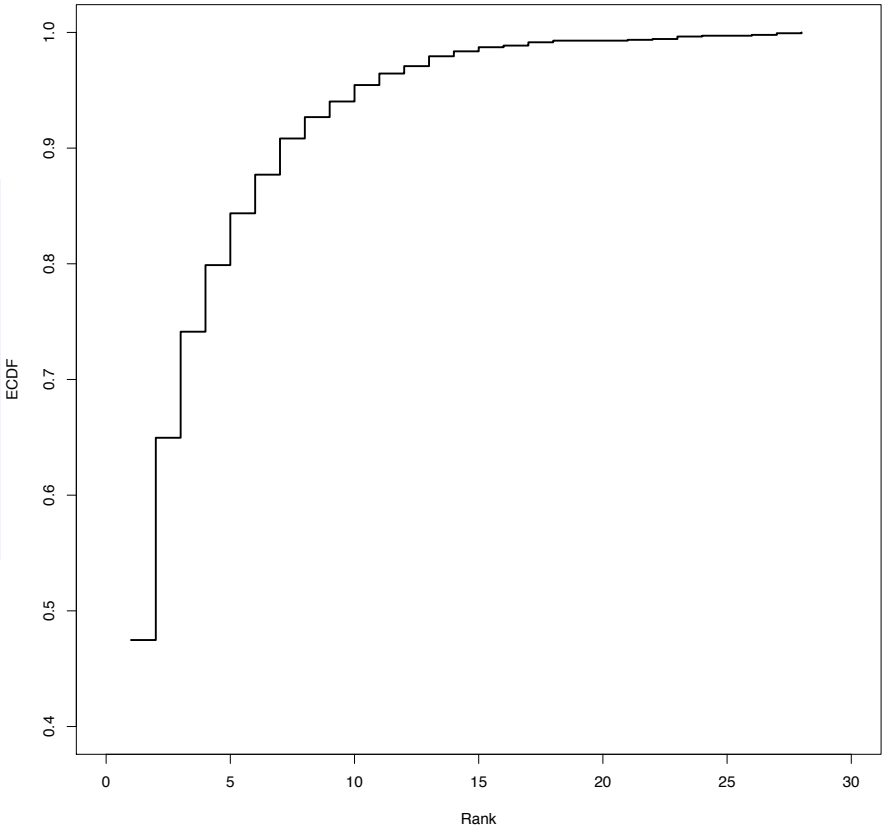
47.47% of the “best” moves are ranked 1 at depth 5

17.48% of the “best” moves are ranked 2 at depth 5

9.17% of the “best” moves are ranked 3 at depth 5

…

0.07% of the “best” moves are ranked 28 at depth 5



Biased Random Choice

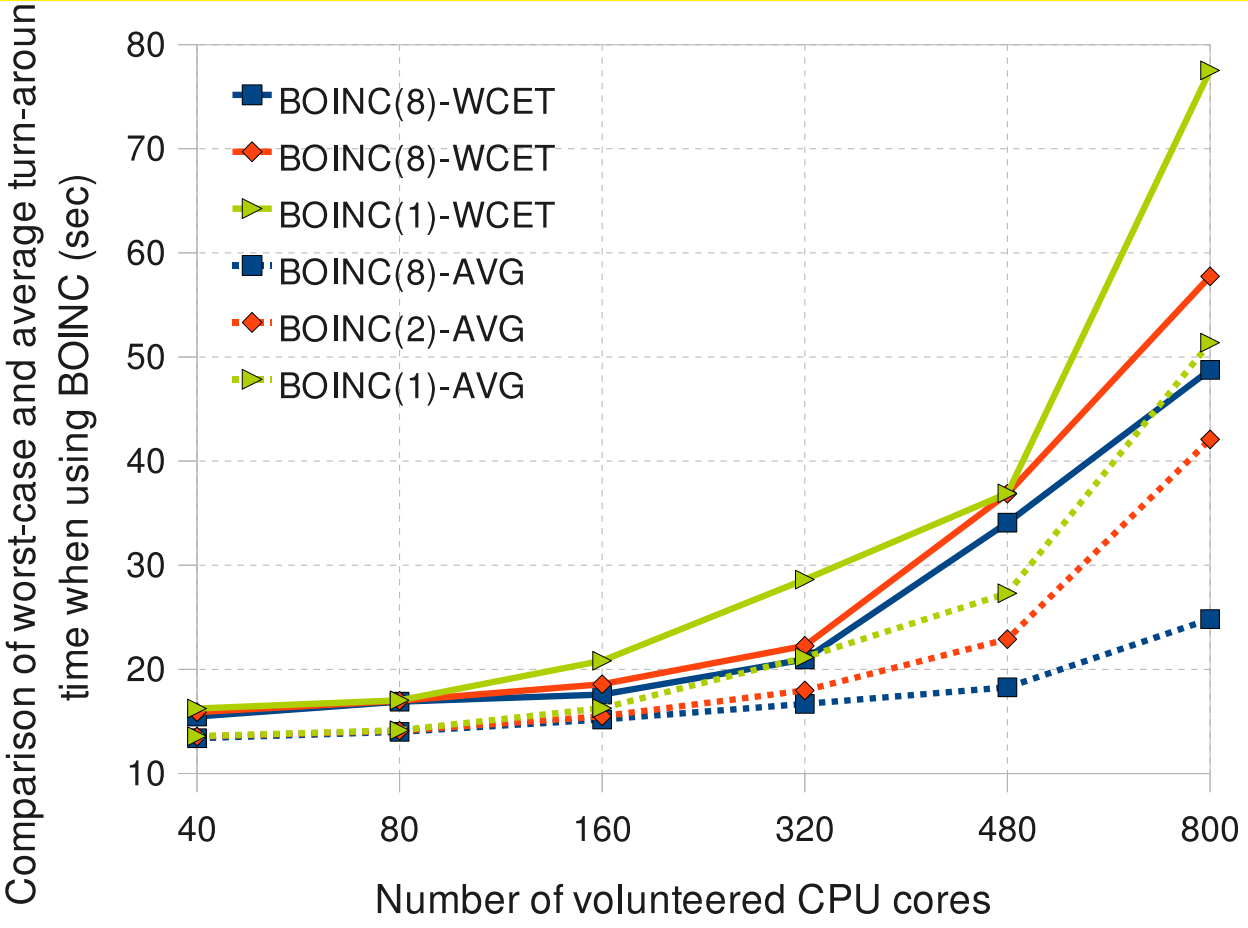
Method:

- For each position
- Enumerate each possible moves
- Rank them according to the depth-5 estimation
- A move is chosen according to the empirical law shown before (N°1 with 47.47%, N°2 with 17.48% etc.)

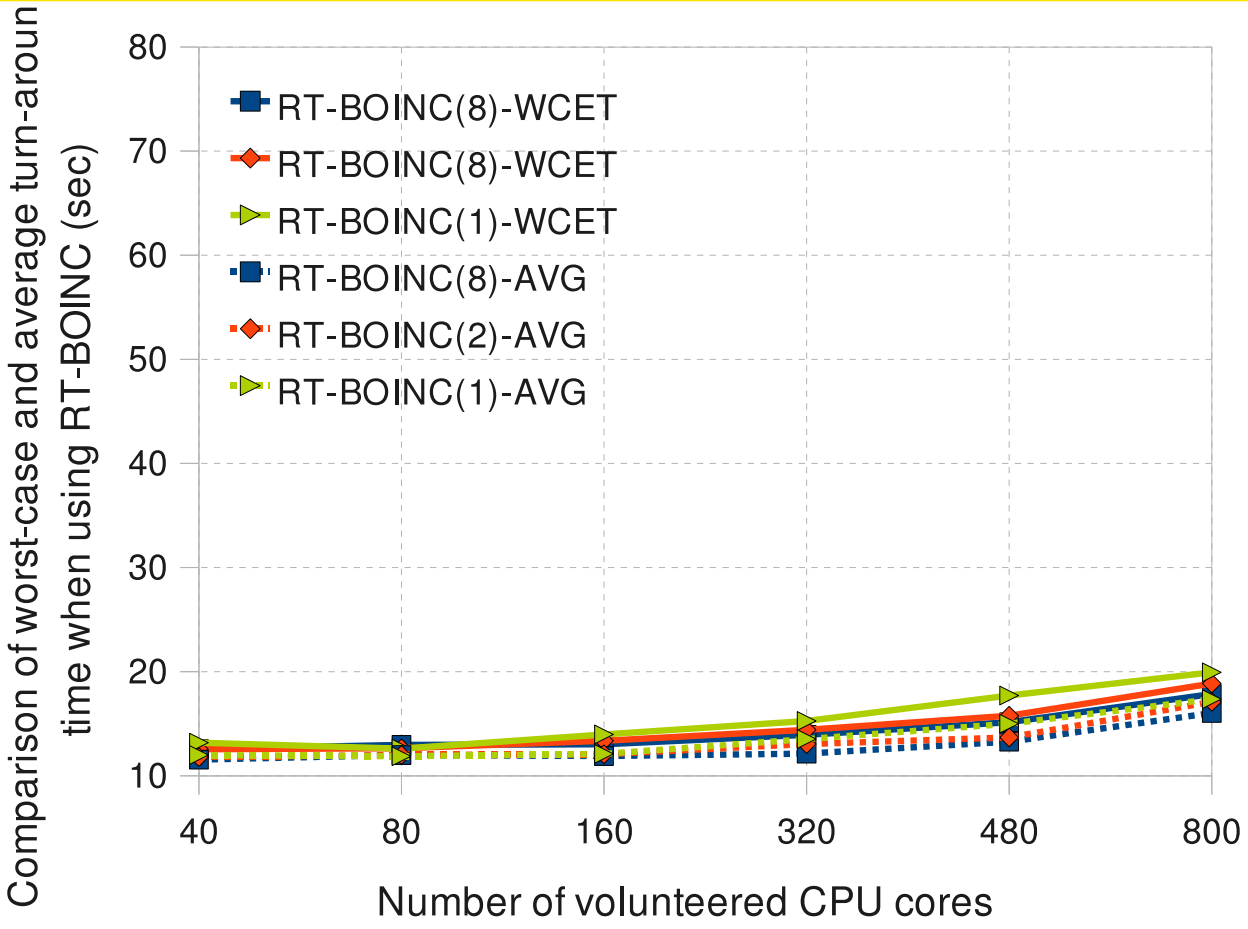
Advantage:

- The more likely a move is to be good the more chance it has to be chosen
- Redundancy of good position (tolerance to churn and failure)

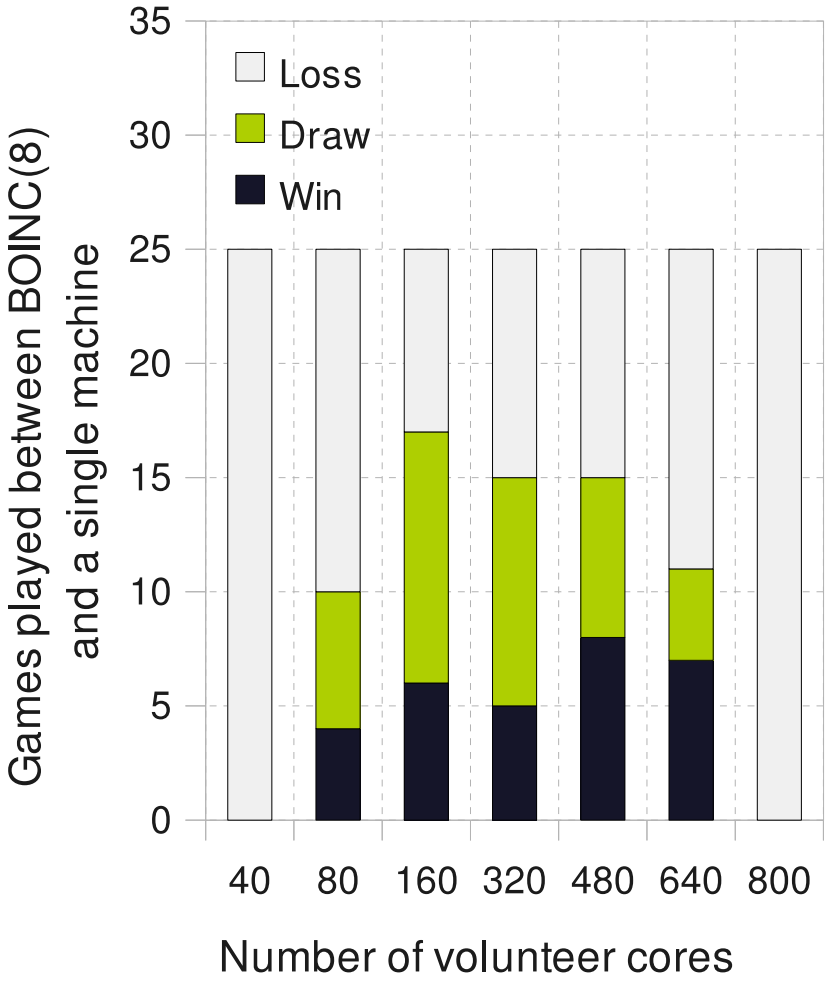
Comparison of worst-case and average time versus # of cores for **BOINC**

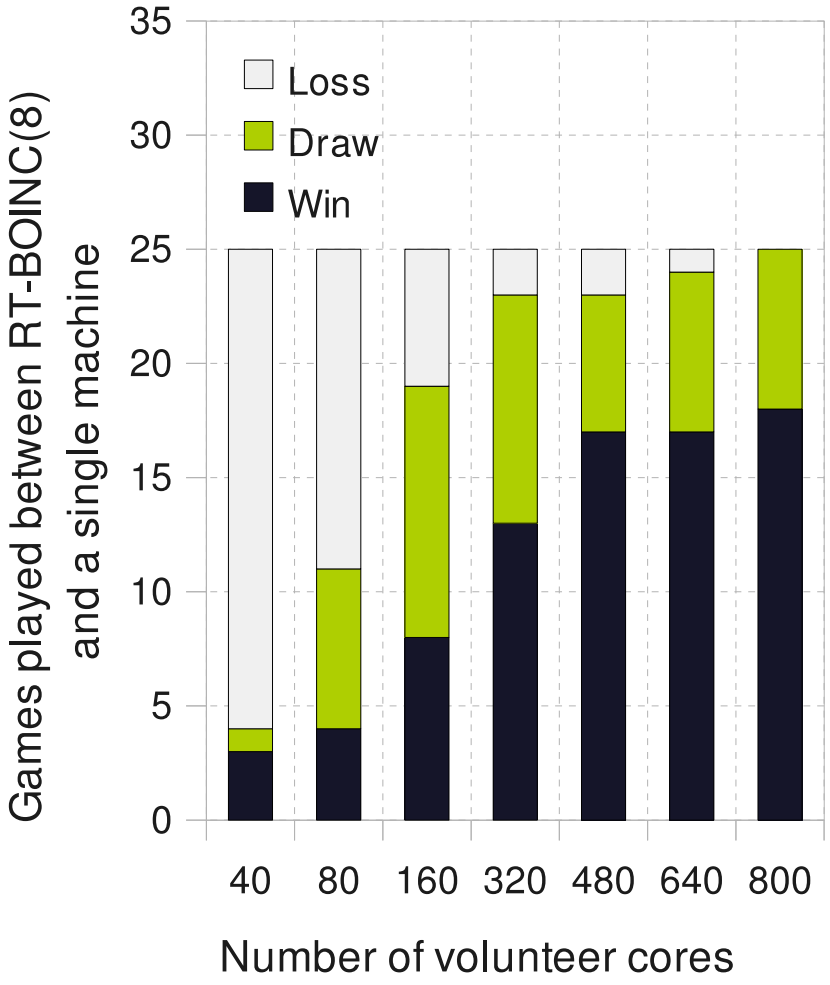


Comparison of worst-case and average time versus # of cores for **RT-BOINC**

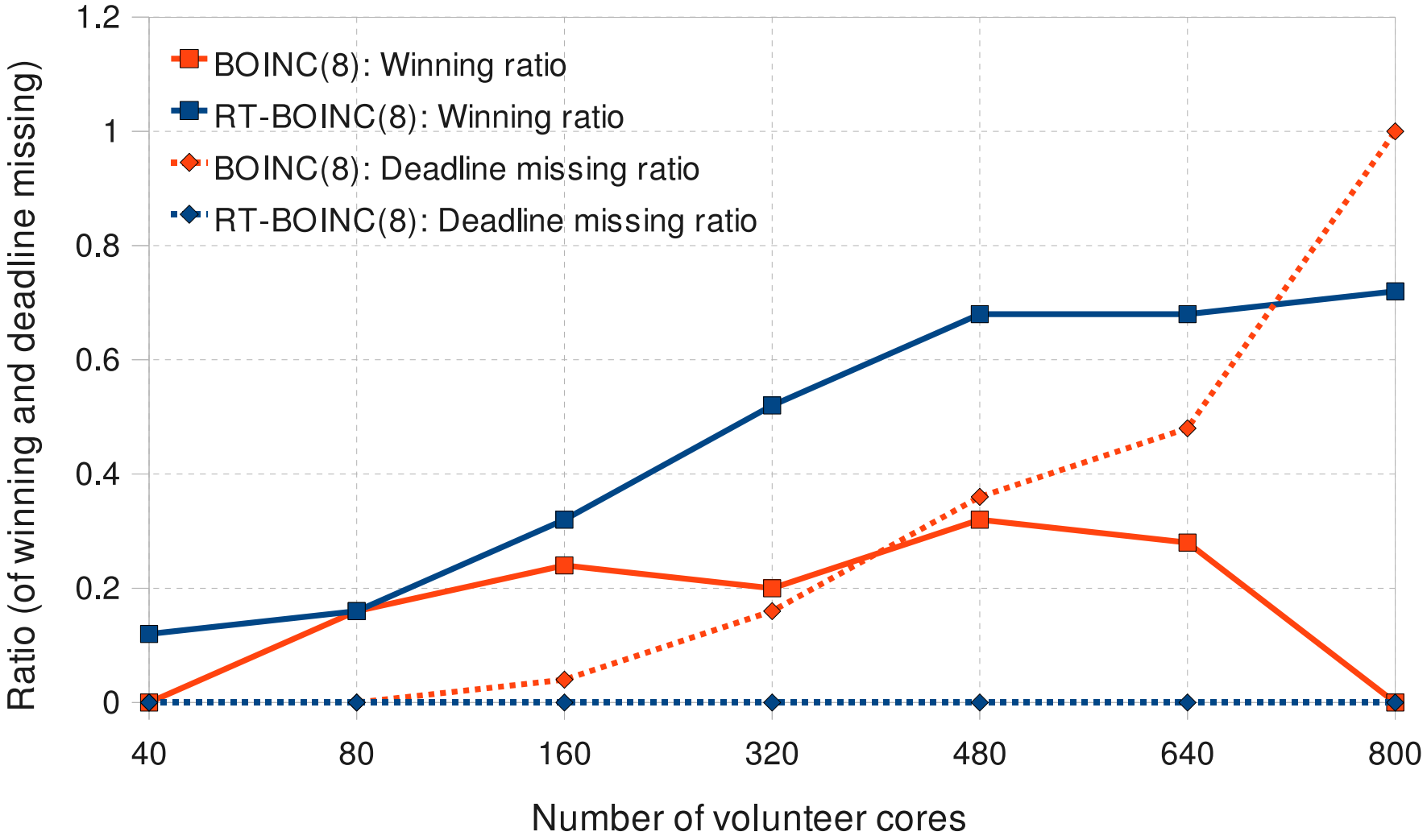


Chess Game results





Winning and Deadline Misses For Chess Games



Summary

- Design and implementation of RT-BOINC
- Performance evaluation with real-time applications: Go and Chess
  - Distributed mechanism for both games
  - Random algorithm on the client side for scalability and reliability
  - Bias random choice based on empirical analysis of positions
- RT-BOINC outperforms BOINC in terms of both average and worst-case response time, deadline miss ratio, and scalability

THANK YOU

BACKUP SLIDES

FAQ

- What about availability and network latency?
  - Plenty of existing work that could be applied and integrated with BOINC
    - Availability: Weissman et al, Kondo et al.
    - Network latency: Stoica et al.
- Why do you care about games?
  - Because millions of people play against these engines

Go Parameters

Parameter	Setting
Board size	9×9
The game of Go server	KGS Go Server
Protocol between players	Go Text Protocol
Player's AI engine	Fuego 0.4.1
Opponent's AI engine	GNU Go 3.8
Computation time for each worker	5 seconds
Deadline for each move	25 seconds