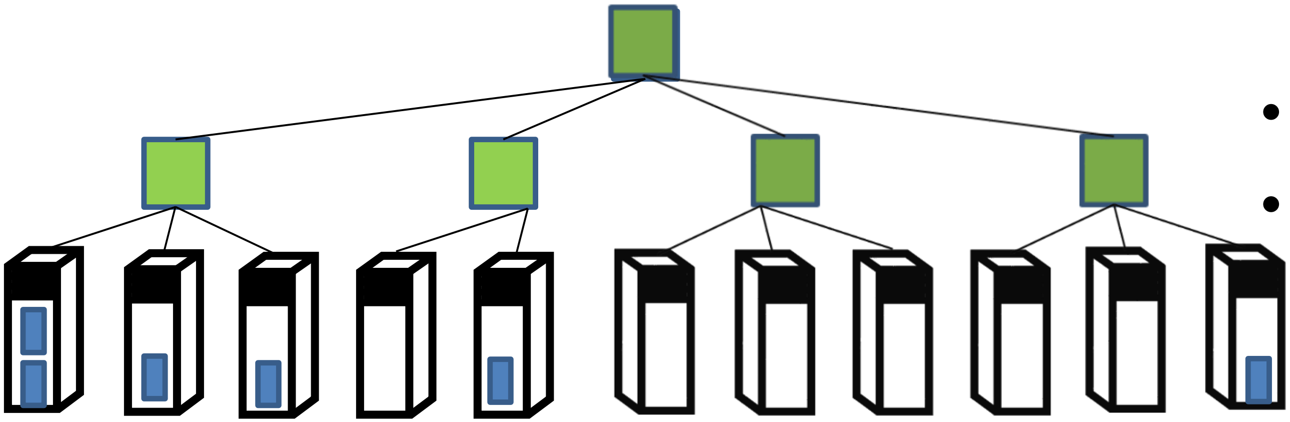
Generalizing Resource Allocation for the Cloud

Anshul Rai, Ranjita Bhagwan, Saikat Guha
Microsoft Research India

Resource Allocation Scenario



- Capacity-based VM Allocation
- Security domains
- Availability domains

Resource Allocation problems keep changing and adapting

*-Allocation

- VM Allocation
- Storage Allocation
- VLAN Allocation
- IP Address Space Allocation
- Server Allocation
- Network Allocation

Current approach

- Resource Management Tools (VMware, Microsoft, etc)
  - Implement their own heuristics
  - Often, not exactly what the administrator needs

- Custom Heuristics
  - Write and test the heuristics code
  - Change the code, repeat testing every time allocation constraints change.
  - Sometimes, constraints start conflicting. Heuristics difficult in such scenarios.

Why not consolidate?

- All these problems are variants of bin-packing
- So why not build a generic resource allocation service
- Reduces the pain of designing, writing, testing and extending custom heuristics

Solver-based Allocation

- Constraint-based programing
  - Z3, Kodkod, eCLiPse
- Built our first version of allocation service
  - Used Z3 and eCLiPse
  - Tough to write constraints
  - Too slow in a number of cases

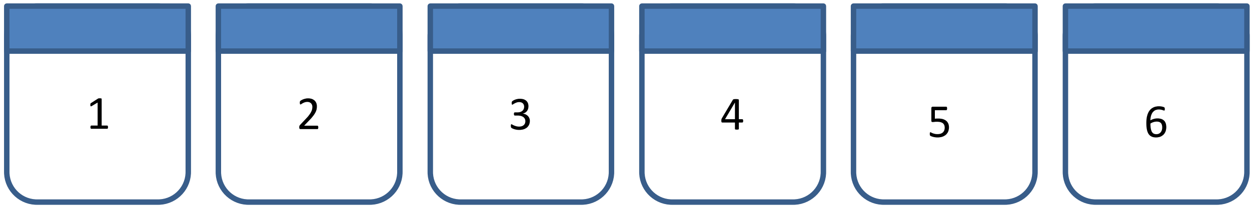
Wrasse
(Resource Allocation Service)

- Tough to write constraints
- **Front End: “Balls and Bins” abstraction**
- Too slow
- **Back End: GPU-based solution generation**

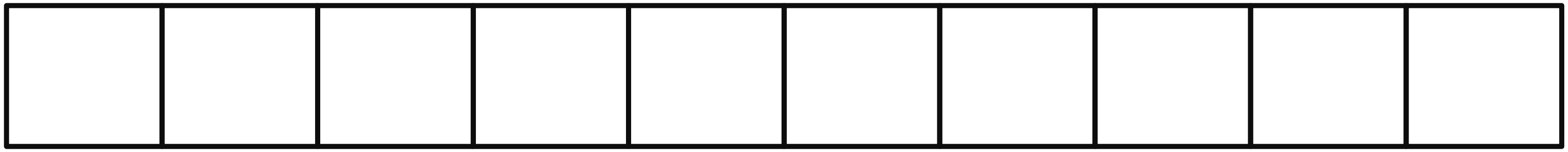
Wrasse Abstraction

BALLS: Virtual Machines

BINS: Servers



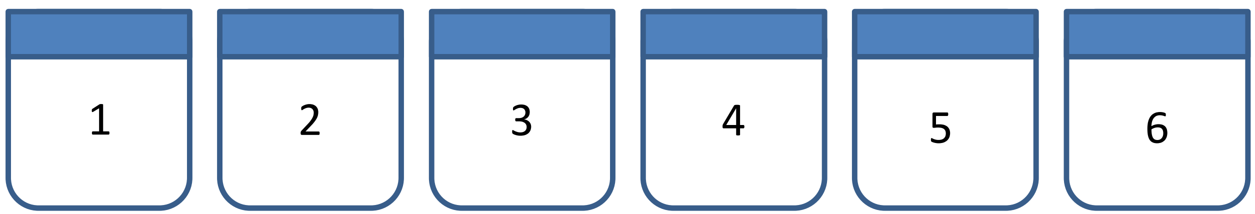
RESOURCES



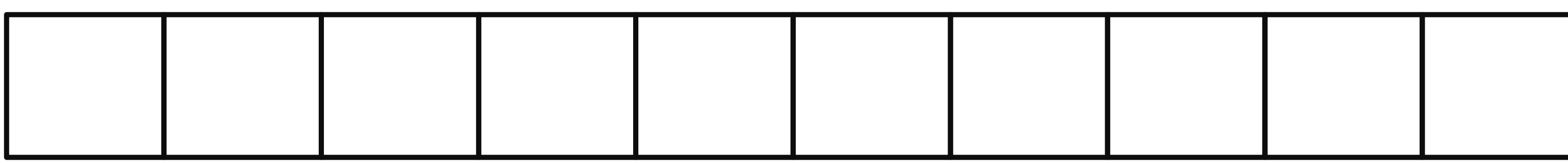
Wrasse Abstraction

BALLS: Virtual Machines

BINS: Servers



RESOURCES



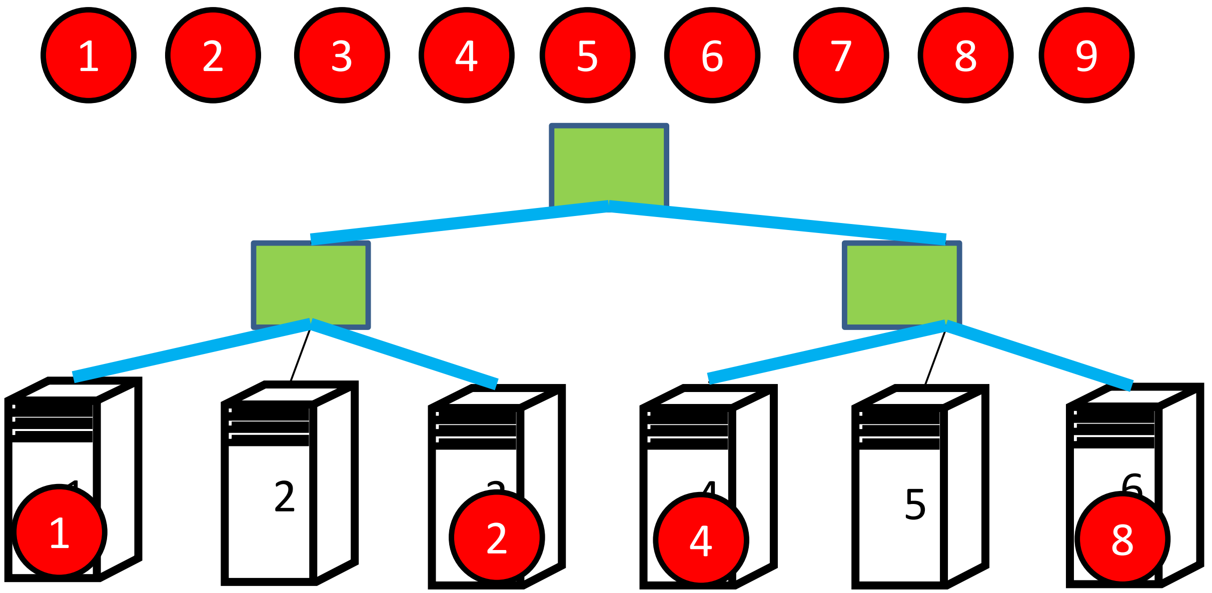
Server 1 CPU capacity

Server 2 CPU capacity

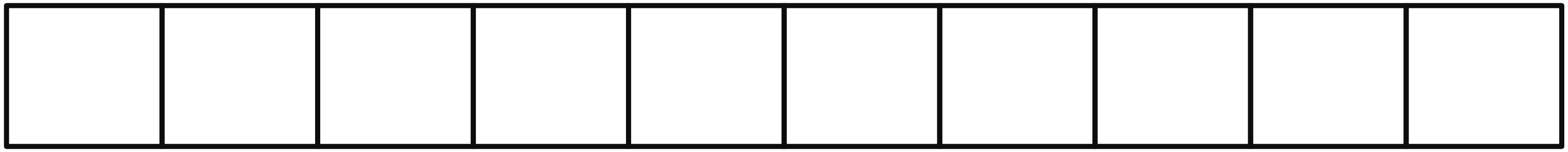
Wrasse Abstraction

BALLS: Virtual Machines

BINS: Servers



RESOURCES



Server 1 CPU capacity

Server 2 CPU capacity

Server 6 CPU capacity

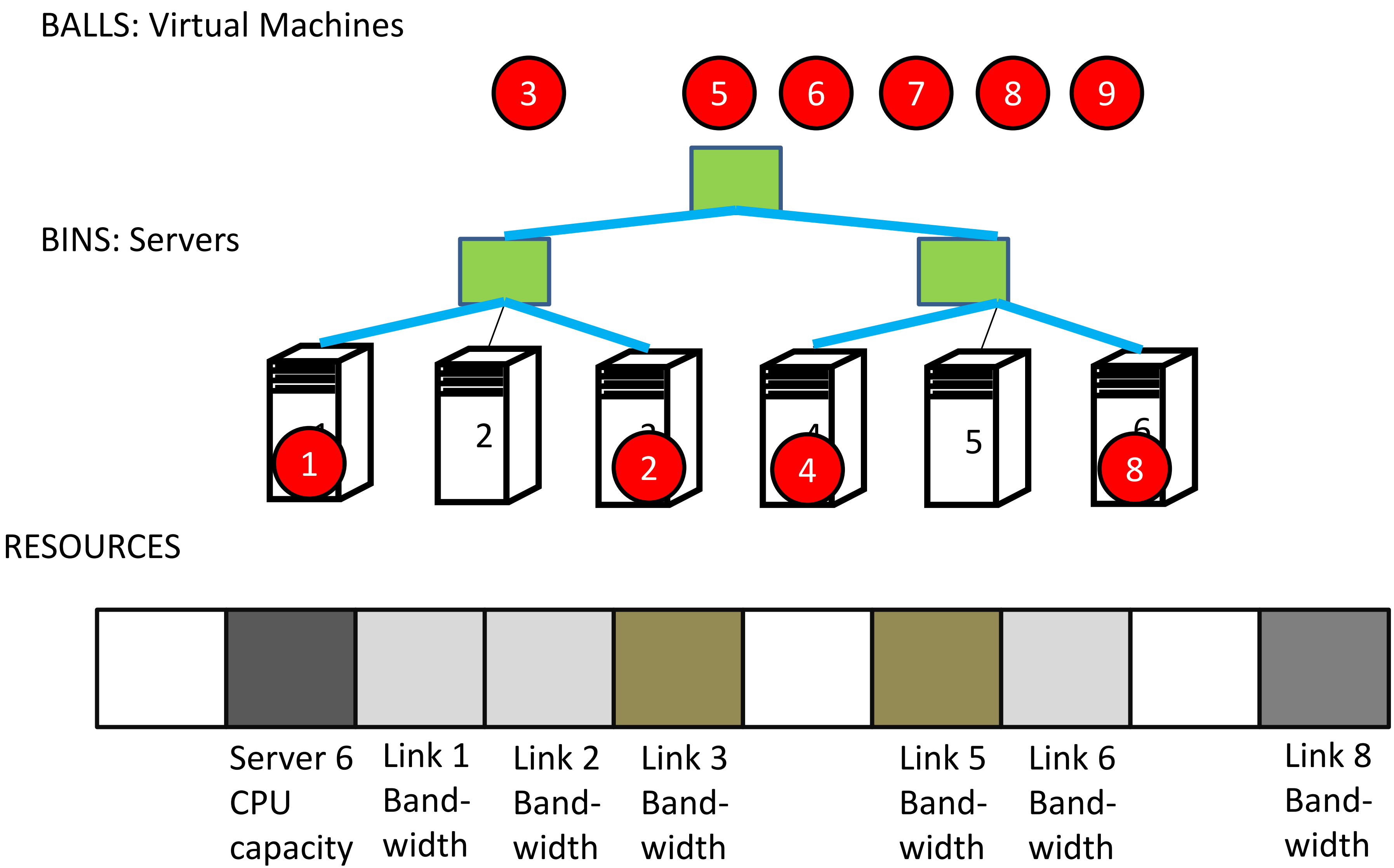
Link 1 Band-width

Link 2 Band-width

Resource Utilization Function

- If Ball X goes into Bin Y, which resources are used, and by how much?
  - Depends on the allocation so far

Resource Utilization Function



Abstraction

- Declare: balls, bins and resources with their capacities
- Write: Resource allocation function.

VM Placement Specification

```
BALLS: {0⇒VM0; 1⇒VM1; 2⇒VM2; 3⇒VM3}
BINS: {0⇒S0; 1⇒S1}
RESOURCES: {0 ⇒ (S0CPU, 100); 1 ⇒ (S0MEM, 5);
            2 ⇒ (S1CPU, 200); 3 ⇒ (S1MEM, 10)}

procedure UTILFN(BALL, BIN, ALLOC)
    UTILDATA: {0 ⇒ 100; 1 ⇒ 2;
               2 ⇒ 50; 3 ⇒ 3;
               4 ⇒ 40; 5 ⇒ 4;
               6 ⇒ 40; 7 ⇒ 4}
    UTIL ← {0, 0, 0, 0}
    UTIL[BIN × 2] ← UTILDATA[BALL × 2]
    UTIL[BIN × 2 + 1] ← UTILDATA[BALL × 2 + 1]
    return UTIL

FOES: [{VM2,VM3}]
```

Friends, Foes and Pinning

- Friends
  - Always put them on the same bin
- Foes
  - Put at least one of the foes in a different bin
- Pin
  - Pin ball X on bin Y
  - Important for incremental changes

Soft constraints

- "Satisfy friend constraint with a probability of 90%"
- "Allow Server 1's CPU capacity to go above limit by 10% with a probability of 5%"

Evolving the Allocation Spec

```
BALLS: {0⇒VM0; 1⇒VM1; 2⇒VM2; 3⇒VM3}
BINS: {0⇒S0; 1⇒S1}
RESOURCES: {0 ⇒ (S0CPU, 100); 1 ⇒ (S0MEM, 5);
            2 ⇒ (S1CPU, 200); 3 ⇒ (S1MEM, 10)}

procedure UTILFN(BALL, BIN, ALLOC)
    UTILDATA: {0 ⇒ 100; 1 ⇒ 2;
               2 ⇒ 50; 3 ⇒ 3;
               4 ⇒ 40; 5 ⇒ 4;
               6 ⇒ 40; 7 ⇒ 4}
    UTIL ← {0, 0, 0, 0}
    UTIL[BIN × 2] ← UTILDATA[BALL × 2]
    UTIL[BIN × 2 + 1] ← UTILDATA[BALL × 2 + 1]
    return UTIL

FOES: [{VM2,VM3}]
```

Evolving the Allocation Spec

SecondNet: Network Virtualization

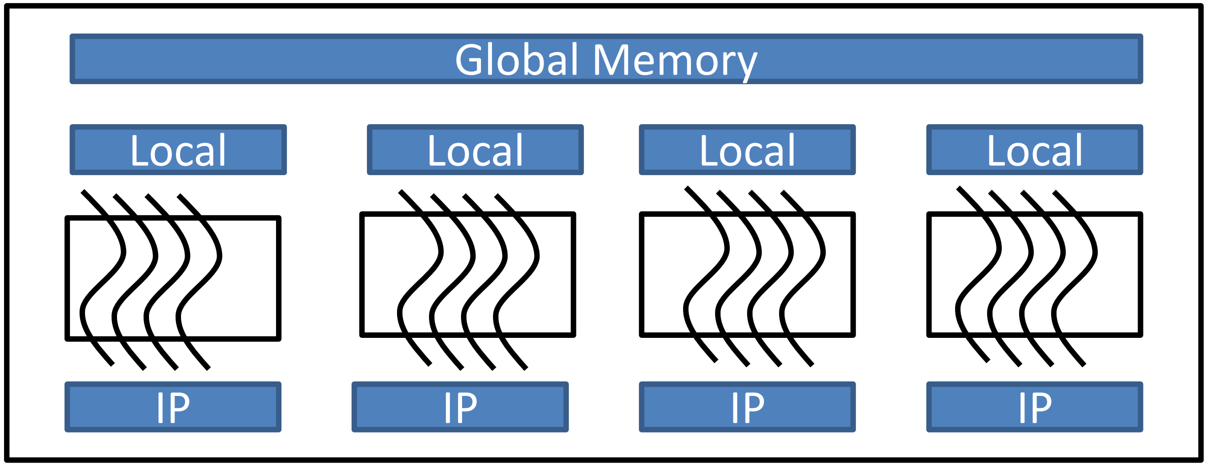
```
...
RESOURCES: {..., 4 ⇒ (LINK0, 150), 5 ⇒ (LINK1, 100)}

procedure UTILFN(BALL, BIN, ALLOC)
    ...
    BW: {0 ⇒ {0, 10, 0, 0};                                  /* VM0 TRAFFIC */
         1 ⇒ {20, 0, 0, 0};                                  /* VM1 TRAFFIC */
         2 ⇒ {0, 0, 0, 50};                                  /* VM2 TRAFFIC */
         3 ⇒ {0, 0, 50, 0}}                                  /* VM3 traffic */
    PATH: {0 ⇒ {1 ⇒ [4, 5]};                                 /* S0 → S1 PATH */
           1 ⇒ {0 ⇒ [5, 4]}}                                 /* S1 → S0 path */
    for all OBALL in 0...3 except BALL do
        OBIN ← ALLOC[OBALL]
        if OBIN ≠ NULL and OBIN ≠ BIN then
            for all LINK in PATHTOLCA[BIN][OBIN] do
                UTIL[LINK] +← BW[BALL][OBALL] + BW[OBALL][BALL]
        if OBIN == NULL then
            for all LINK in PATHTOROOT[BIN] do
                UTIL[LINK] +← BW[BALL][OBALL] + BW[OBALL][BALL]
        if OBIN ≠ NULL and OBIN == BIN then
            for all LINK in PATHTOROOT[BIN] do
                UTIL[LINK] -← BW[BALL][OBALL] + BW[OBALL][BALL]
    ...
```

A Discussion on this Design

- Balls of only one type, bins of only one type
- No notion of a network
- As a result, resource utilization function can get complicated
- But simplicity important for solver implementation.

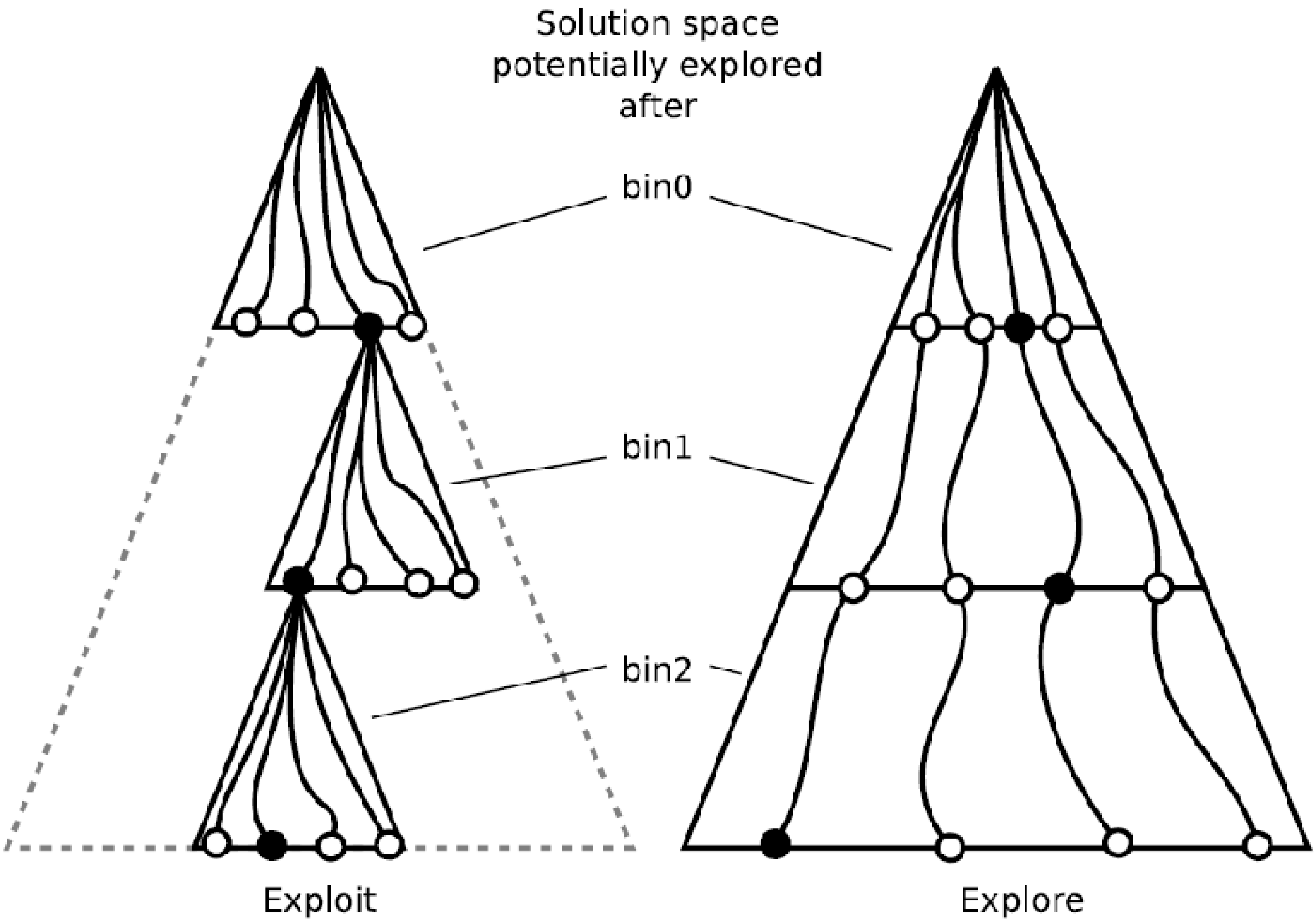
Can we model different kinds of balls?
Can we model different kinds of bins?
Can we model resource utilizations other than additive?

Back End: GPU-based solver



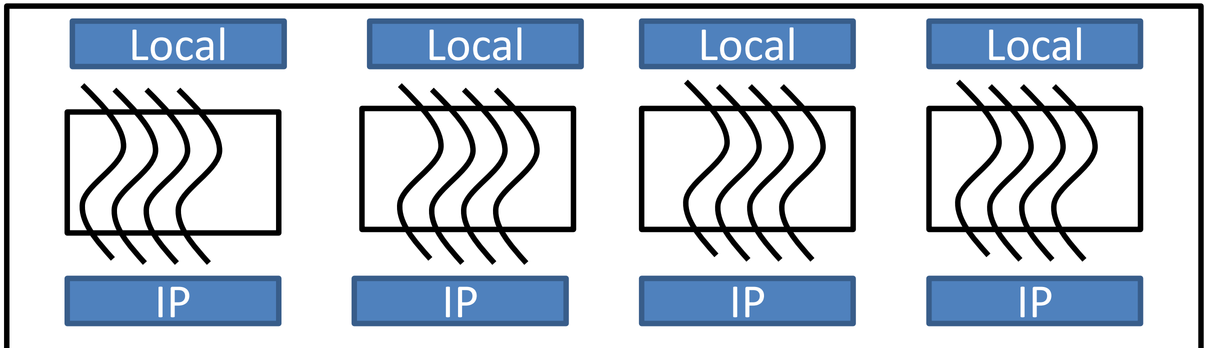
- Pick a ball at random
- Put it in the first bin
- Satisfy all Friend-Foe constraints
- Use resource utilization funtion to ensure no resource capacities are exceeded
- Pick another ball … until all balls have been tried for this bin.

Explore vs Exploit



GPU Implementation

- Version 1: Each thread finds a potential solution (16 solutions simultaneously checked)
  - Memory issues
  - Scale issues
- Version 2: Each thread-group finds a potential solution (4 solutions simultaneously checked)



VM Placement

Input

Application	VMs	Avg. CPU (Fraction of Total CPU)	Avg. Memory (GB)	Avg. Disk (MBps)	Av. Network out (MBps)	Avg. Network in (MBps)
PgRank	474	0.16	2.94	7.67	1.95	2.03
ClkBot	885	0.14	1.07	19.69	0.78	1.22
ImgProc	2942	0.37	0.35	1.41	0.92	0.04

Solution quality (comparing to SCVMM heuristics)

Application	FFDProd	DotProd	NBG	Z3	Wrasse
PgRank	90	100	97	90	89
ClkBot	420	420	420	424	420
ImgProc	1406	1403	1406	1417	1403

Solution time (ms)

Application	FFDProd	DotProd	NBG	Z3	Wrasse
PgRank	7	16.2	19.2	30864	51
ClkBot	15.6	69.2	82.6	146149	7645
ImgProc	92.6	744.2	923.6	139876	370

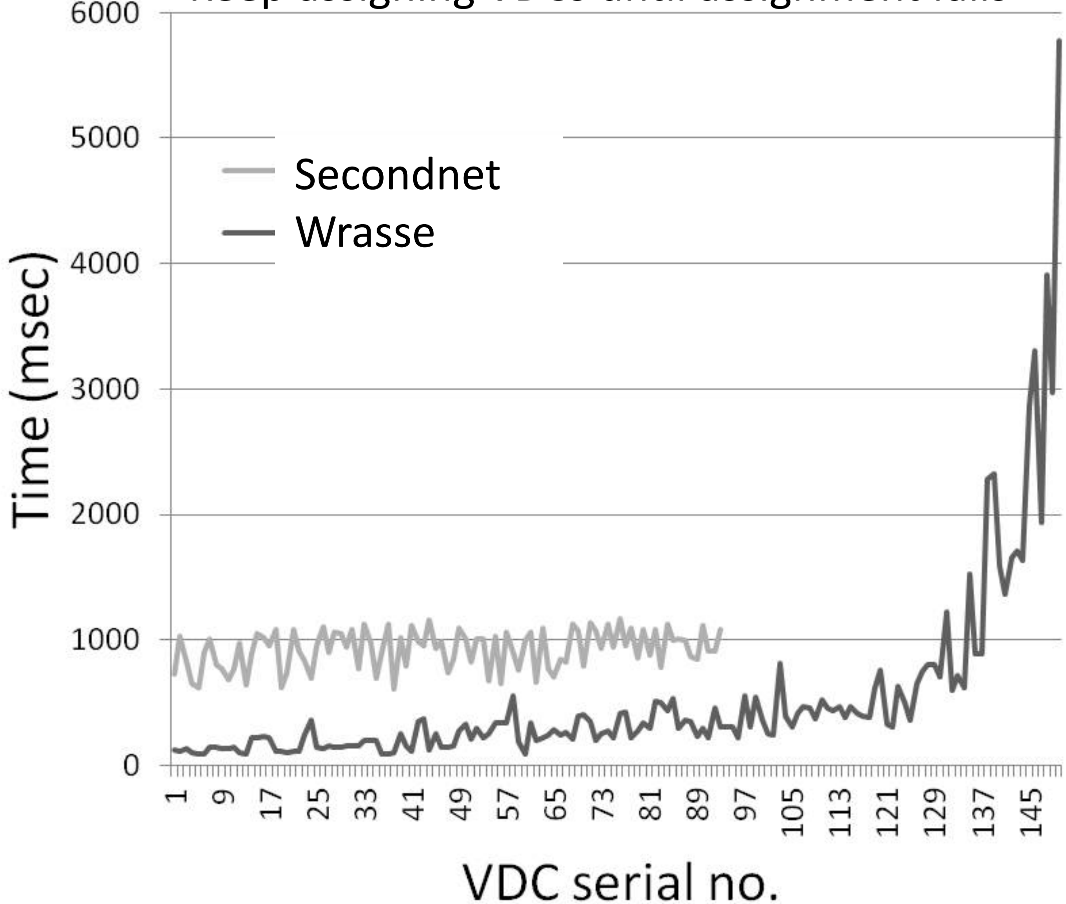
Network Virtualization

SecondNet (CoNext 2010)

1024 servers, 2-level fat-tree.

Average Virtual Data Center (VDC) size: 94.

Keep assigning VDCs until assignment fails



Performance: GPU vs CPU

- Used AMD HD6990 and the nVidia Tesla
- Tesla implementation worked about 8.5 times faster than 3 GHz Intel Core 2 Duo processor

Related Work

- Rhizoma: Used eCLiPse for configuration management
  - Runs into performance issues with large-sized problems.
- Cologne: Distributed platform for configuration management
  - Uses constraint solvers as well in the back-end.

- Various heuristic-based solutions for configuration mangement
  - Wrasse can encode all that we have encountered.

Summary

- Presented a generic resource allocation service for the cloud
- Good performance, both in terms of time to run and solution quality
- We have built a web service around Wrasse so it can be easily used

Questions?