

# Distributed solving through model splitting

**Lars Kotthoff**, Neil Moore  
University of St Andrews

06 September 2010

# Motivation

**semigroup** set with closed binary operator

**order  $n$  semigroup** set of size  $n$

# Motivation

**semigroup** set with closed binary operator

**order  $n$  semigroup** set of size  $n$

$\circ$	1	2	3	...
1				
2				
3				
...				

# Motivation

**semigroup** set with closed binary operator

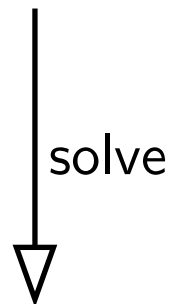
**order  $n$  semigroup** set of size  $n$

$\circ$	1	2	3	...
1				
2				
3				
...				

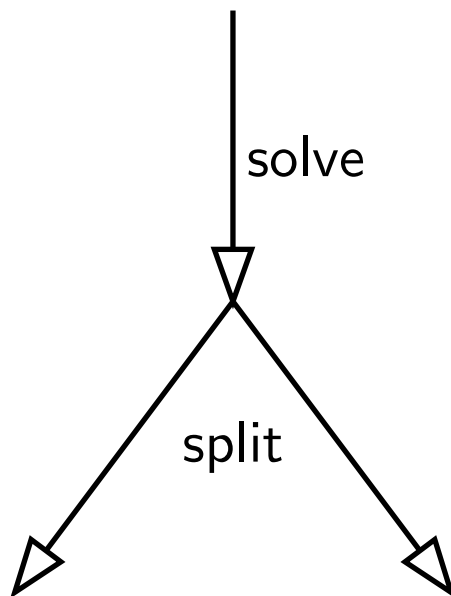
**order 9**  $\approx$  178 CPU hours

**order 10** 200 CPU years (estimated)

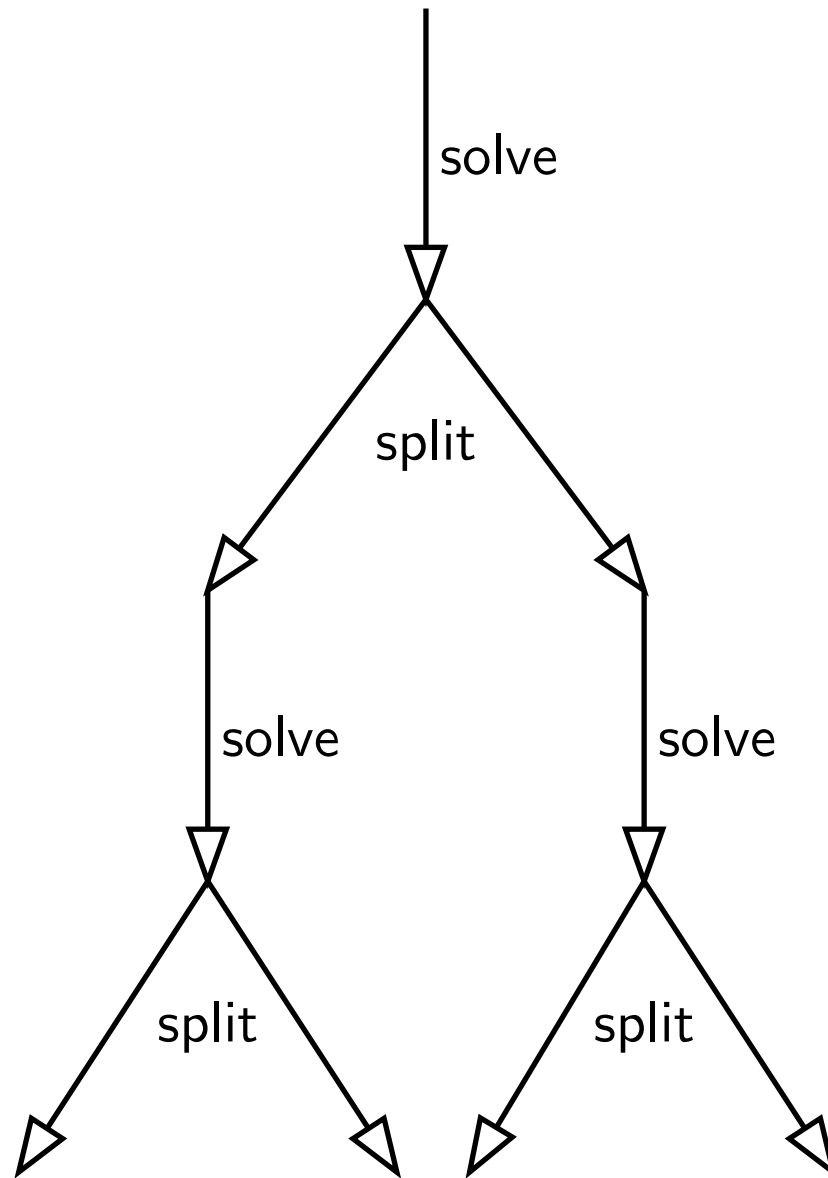
# Approach



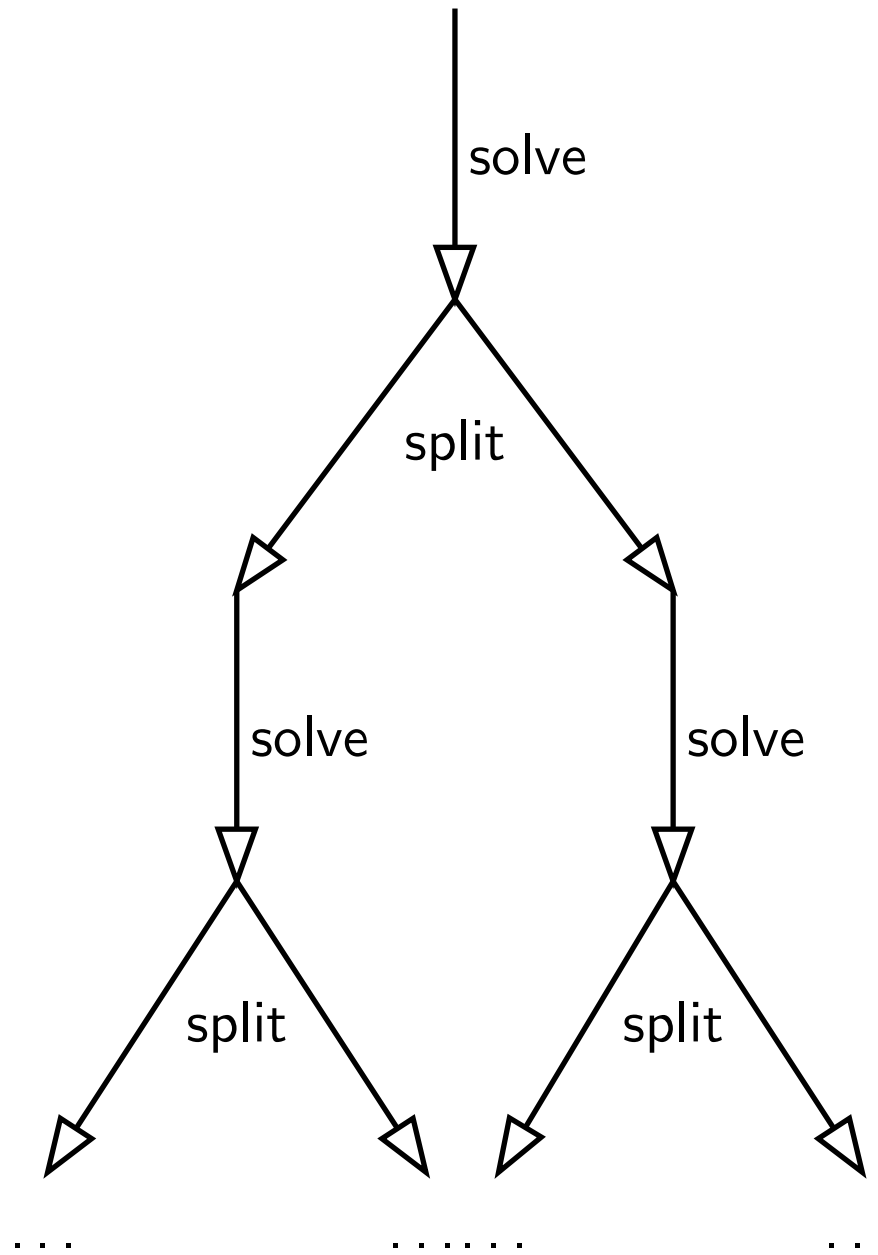
# Approach



# Approach



# Approach



# Minion

```
1 diff --git a/minion/BuildCSP.cpp b/minion/BuildCSP.cpp
2 index b1c62ff..2e2e2cf 100644
3 --- a/minion/BuildCSP.cpp
4 +++ b/minion/BuildCSP.cpp
5 @@ -45,10 +45,9 @@ void BuildCSP(StateObj* stateObj, CSPInstance* instance)
6 {
7     print_matrix.push_back(BuildCon: get_AnyVarRef_from_Var(stateObj, instance, print_matrix));
8 }
9 // Impose Constraints
10 while(instance.constraints.empty())
11 {
12     getState(stateObj).addConstraint(build_constraint(stateObj, instance.constraints.front());
13     instance.constraints.pop_front();
14     for(list<ConstraintBlob>::iterator it = instance.constraints.begin(); it != instance.constraints.end(); it++)
15         getState(stateObj).addConstraint(build_constraint(stateObj, *it));
16 }
17 // Solve!
18 diff --git a/minion/MILtools/print_CSP.h b/minion/MILtools/print_CSP.h
19 index 5d6d219..cb0957d 100644
20 --- a/minion/MILtools/print_CSP.h
21 +++ b/minion/MILtools/print_CSP.h
22 @@ -249,10 +249,13 @@ void print_search_info(const vector<Var*& var_vec)
23 {
24 }
25 void build_instance()
26 { build_instance(csp.constraints, csp.vars.get_all_vars()); }
27 { build_instance(csp.constraints, csp.vars.get_all_vars(), true); }
28 }
29 void build_instance(bool printEof)
30 { build_instance(csp.constraints, csp.vars.get_all_vars(), printEof); }
31 }
32 void build_instance(const vector<Var*& varlist_vec)
33 {
34 void build_instance(const vector<Var*& varlist_vec, bool printEof)
35 {
36     list<ConstraintBlob> new_constraint_list;
37     @ -262,11 +262,11 @@ void build_instance(const vector<Var*& varlist_vec)
38     new_constraint_list.push_back(Var);
39 }
40 }
41 build_instance(new_constraint_list, varlist_vec);
42 build_instance(new_constraint_list, varlist_vec, printEof);
43 }
44 void build_instance(const list<ConstraintBlob> constraints,
45     const vector<Var*& varlist)
46 {
47     const vector<Var*& varlist, bool printEof)
48 {
49     oss << "MINION 2" << endl;
50 }
51 @ -287,7 +290,7 @@ void build_instance(const list<ConstraintBlob> constraints)
52 {
53     print_instance(Var);
54 }
55 oss << "***EOF***" << endl;
56 if(printEof) oss << "***EOF***" << endl;
57 }
58 }
59 diff --git a/minion/command_line_parse.cpp b/minion/command_line_parse.cpp
60 index a3c0044..244e06e 100644
61 --- a/minion/command_line_parse.cpp
62 +++ b/minion/command_line_parse.cpp
63 @@ -275,6 +275,10 @@ void parse_command_line(StateObj* stateObj, SearchMethod& args, int argc, char**
64 {
65     getOptions(stateObj).noresumefile = true;
66 }
67 else if(command == string("-split"))
68 {
69     getOptions(stateObj).split = true;
70 }
71 else if(command[0] == '-' && command != string("-"))
72 {
73     cout << "I don't understand " << command << ". Sorry." << endl;
74 }
75 diff --git a/minion/search/common_search.h b/minion/search/common_search.h
76 index 59c05a3..9d8809e 100644
77 --- a/minion/search/common_search.h
78 +++ b/minion/search/common_search.h
79 @@ -25,6 +29,7 @@ namespace Controller
80 {
81     getOptions(stateObj).findAllSolutions();
82     getState(stateObj).setOptimizeVar(new AnyVarRef(var));
83     getState(stateObj).setOptimizeProblem(true);
84     getState(stateObj).setMaximize(true);
85 }
86 // Sets optimization variable.
87 @ -48,11 +48,11 @@ namespace Controller
88 {
89     getOptions(stateObj).findAllSolutions();
90     getState(stateObj).setOptimizeVar(new AnyVarRef(Var.Neg<VarRef>(var)));
91     getState(stateObj).setOptimizeProblem(true);
92     getState(stateObj).setMaximize(false);
93 }
94 }
95 // Ensures a particular constraint is satisfied by the solution.
96 @ -156,6 +159,7 @@ namespace Controller
97 {
98 #include <fstream>
99 #include ".../MILtools/print_CSP.h"
100 }
101
```

```
102 //repeat declaration
103 struct triple {
104     @ -176,6 +174,6 @@ namespace Controller
105     if(getOptions(stateObj).noresumefile) {
106         return;
107     }
108     string filename = string("minion-resume-") + to_string(getpid());
109     cout << "Output resume file to '" << filename << "' << endl;
110     ofstream fileout(filename.c_str());
111     fileout << "MINION 2" << endl;
112     fileout << "***CONSTRAINTS***" << endl;
113     vector<triple> left_branches_so_far;
114     left_branches_so_far.reserve(branches.size());
115     for(vector<triple>::const_iterator curr = branches.begin(); curr != branches.end(); curr++) {
116         if(curr->isLeft() {
117             left_branches_so_far.push_back(*curr);
118         } else {
119             fileout << "watched-or (" << endl;
120             for(vector<triple>::const_iterator lb = left_branches_so_far.begin();
121                 lb != left_branches_so_far.end();
122                 lb++) {
123                 fileout << "w-notliteral (" << endl;
124                 inputPrint(fileout, stateObj, var_array[lb->var].getBaseVar());
125                 fileout << " " << lb->val << " " << endl;
126             }
127             vector<string> splits;
128             string curvar = "(no split variable)";
129             string opt = "";
130         }
131         if(getOptions(stateObj).split)
132         {
133             if(getState(stateObj).isOptimizationProblem() {
134                 AnyVarRef* optVarRef = getState(stateObj).getOptimizeVar();
135                 string optVar = getState(stateObj).getInstance()->vars.getName(optVarRef->getBaseVar());
136                 Dominant optimal = getState(stateObj).getOptimizeValue();
137                 opt = "ineq(" << endl;
138                 if(getState(stateObj).isMaximize() {
139                     opt = to_string(optimal) + string(", ") + optVar + string(", 0)\n");
140                 } else {
141                     opt = optVar + string(", ") + to_string(-optimal) + string(", 0)\n");
142                 }
143             }
144             fileout << "w-notliteral (" << endl;
145             inputPrint(fileout, stateObj, var_array[curr->var].getBaseVar());
146             fileout << " " << curr->val << " " << endl;
147         }
148         if(branches.empty())
149         {
150             // TODO: We should check if any variable has non-empty domain, but this will do for now.
151             // The most likely case is we have just caught the end of search.
152             splits.push_back("");
153         }
154         else
155         {
156             typedef typename VarArray::value_type VarRef;
157             VarRef& var = var_array[branches.back().var];
158             curvar = getState(stateObj).getInstance()->vars.getName(var, getBaseVar());
159             Dominant min = var.getMax();
160             Dominant max = var.getMin();
161             int med = (min+max)/2;
162             string left("ineq(" << endl;
163             left = curvar + string(", ") + to_string(med) + string(", 0)\n");
164             splits.push_back(left + opt);
165         }
166         string right("ineq(" << endl;
167         right = to_string(med) + string(", ") << endl;
168         right = curvar + string(", -1)\n");
169         splits.push_back(right + opt);
170     }
171 }
172 else
173 {
174     splits.push_back("");
175 }
176 }
177 }
178 }
179 }
180 }
181 int i = 0;
182 for(vector<string>::iterator s = splits.begin(); s != splits.end(); s++) {
183     string basename = getOptions(stateObj).instance_name;
184     size_t npos = basename.find("minion");
185     size_t rpos = basename.find("-resume-");
186     if(rpos != string::npos) {
187         basename = basename.substr(0, rpos);
188     } else if(npos != string::npos) {
189         basename = basename.substr(0, npos);
190     }
191     string filename = basename + "-resume-" + to_string(time(NULL)) + "-" + to_string(getpid()) + "-" + curvar + "-" + to_string(i++) + ".minion";
192     cout << "Output resume file to '" << filename << "' << endl;
193     ofstream fileout(filename.c_str());
194     fileout << "# original instance: " << getOptions(stateObj).instance_name << endl;
195     fileout << "\n";
196     fileout << "#";
197     vector<triple> left_branches_so_far;
198     left_branches_so_far.reserve(branches.size());
199     for(vector<triple>::const_iterator curr = branches.begin(); curr != branches.end(); curr++) {
200         if(curr->isLeft() {
201             left_branches_so_far.push_back(*curr);
202         } else {
203             fileout << "watched-or (" << endl;
204             for(vector<triple>::const_iterator lb = left_branches_so_far.begin();
205                 lb != left_branches_so_far.end();
206                 lb++) {
207                 fileout << "w-notliteral (" << endl;
208                 inputPrint(fileout, stateObj, var_array[lb->var].getBaseVar());
209                 fileout << " " << lb->val << " " << endl;
210             }
211             vector<string> splits;
212             string curvar = "(no split variable)";
213             string opt = "";
214         }
215         if(getOptions(stateObj).split)
216         {
217             if(getState(stateObj).isOptimizationProblem() {
218                 AnyVarRef* optVarRef = getState(stateObj).getOptimizeVar();
219                 string optVar = getState(stateObj).getInstance()->vars.getName(optVarRef->getBaseVar());
220                 Dominant optimal = getState(stateObj).getOptimizeValue();
221                 opt = "ineq(" << endl;
222                 if(getState(stateObj).isMaximize() {
223                     opt = to_string(optimal) + string(", ") + optVar + string(", 0)\n");
224                 } else {
225                     opt = optVar + string(", ") + to_string(-optimal) + string(", 0)\n");
226                 }
227             }
228             fileout << "w-notliteral (" << endl;
229             inputPrint(fileout, stateObj, var_array[curr->var].getBaseVar());
230             fileout << " " << curr->val << " " << endl;
231         }
232         if(branches.empty())
233         {
234             // TODO: We should check if any variable has non-empty domain, but this will do for now.
235             // The most likely case is we have just caught the end of search.
236             splits.push_back("");
237         }
238         else
239         {
240             typedef typename VarArray::value_type VarRef;
241             VarRef& var = var_array[branches.back().var];
242             curvar = getState(stateObj).getInstance()->vars.getName(var, getBaseVar());
243             Dominant min = var.getMax();
244             Dominant max = var.getMin();
245             int med = (min+max)/2;
246             string left("ineq(" << endl;
247             left = curvar + string(", ") + to_string(med) + string(", 0)\n");
248             splits.push_back(left + opt);
249         }
250         string right("ineq(" << endl;
251         right = to_string(med) + string(", ") << endl;
252         right = curvar + string(", -1)\n");
253         splits.push_back(right + opt);
254     }
255 }
256 // The format of output used (-1 for default)
257 int outputType;
258 }
259 @ -265,7 +272,7 @@ public:
260 time_limit_is_CPU_time(false), parser_verbose(false),
261 randomize_valvar_order(false), redump(false),
262 redump(false), graph(false), instance_stats(false),
263 resume(false), noresumefile(false),
264 resume(false), noresumefile(false), split(false),
265 outputType(-1), noTimers(false)
266 }
267
```

```
204 + for(vector<triple>::const_iterator lb = left_branches_so_far.begin();
205 +     lb != left_branches_so_far.end();
206 +     lb++) {
207 +     fileout << "w-notliteral (" << endl;
208 +     inputPrint(fileout, stateObj, var_array[lb->var].getBaseVar());
209 +     fileout << " " << lb->val << " " << endl;
210 + }
211 + fileout << "w-notliteral (" << endl;
212 + inputPrint(fileout, stateObj, var_array[curr->var].getBaseVar());
213 + fileout << " " << curr->val << " " << endl;
214 + }
215 + fileout << "***EOF***" << endl;
216 + }
217 + fileout << "***EOF***" << endl;
218 + }
219 + }
220 }
221 // Check if timelimit has been exceeded.
222 diff --git a/minion/solver.h b/minion/solver.h
223 index 617addb..b3d616f 100644
224 --- a/minion/solver.h
225 +++ b/minion/solver.h
226 @ -44,6 +44,7 @@ class SearchState
227 AnyVarRef* optimize_var;
228 Dominant current_optimize_position;
229 bool optimize;
230 bool maximize;
231 }
232 // The variables to print when a solution is found.
233 vector<vector<AnyVarRef> > print_matrix;
234 @ -98,10 +99,13 @@ public:
235 Dominant getOptimizeValue() { return current_optimize_position; }
236 void setOptimizeValue(Dominant optimize_pos) { current_optimize_position = optimize_pos; }
237 bool isOptimizationProblem() { return optimize; }
238 void setOptimizationProblem(bool optimize) { optimize = _optimize; }
239 bool isMaximize() { return maxime; }
240 void setMaximize(bool maxime) { maxime = _maximize; }
241 void addConstraint(AbstractConstraint& c);
242 vector<AbstractConstraint*& getConstraintList() { return constraints; }
243 @ -246,6 +250,9 @@ public:
244 // Do not write a resume file.
245 bool noresumefile;
246 // split search tree in half on time out
247 bool split;
248 // The format of output used (-1 for default)
249 int outputType;
250 @ -265,7 +272,7 @@ public:
251 time_limit_is_CPU_time(false), parser_verbose(false),
252 randomize_valvar_order(false), redump(false),
253 redump(false), graph(false), instance_stats(false),
254 resume(false), noresumefile(false),
255 resume(false), noresumefile(false), split(false),
256 outputType(-1), noTimers(false)
257 }
```

# Control

```
#!/usr/bin/env ruby
require 'inotify'

Thread.abort_on_exception = true

POLL = 3600
MACHINES = [ ]
BIN = "/tmp/minion-static"
LIMIT = 3600
ARGS = "-findallsols -noprintsols -split -searchlimit " + LIMIT.to_s

MAXQUEUE = ARGV[0].to_i
queued = 0

torun = []

def log(s)
  puts Time.new.strftime("%m/%d %H:%M:%S ") + s
end

def queue(f)
  tries = 0
  condor = [ "Universe = Vanilla",
            "should_transfer_files = YES",
            "when_to_transfer_output = ON_EXIT",
            ("requirements = (Arch==\`X86_64\` || Arch==\`INTEL\`) && (" +
             MACHINES.collect { |m|
               "machine != \`${m}\`"
             }.join(" && ") + ")"),
            "Notification = Never",
            "transfer_executable = False",
            "Executable = " + BIN,
            "input = " + f,
            "arguments = \`${ARGS}\` " + f + "\`,",
            "output = " + f + ".out",
            "Queue 1" ].join("\n")

  begin
    tries += 1
    cluster = IO.popen("condor_submit", "r+") { |io|
      io.write condor
      io.close_write
      (io.read[/[0-9]+\./])[0..-2]
    }
    log "submitted " + f + ", cluster " + cluster
  rescue
    if tries <= 3 then
      retry
    else
      log "*** Failed to queue " + f
    end
  end
end

end

i = Inotify.new
t = Thread.new do
  i.each_event do |ev|
    #puts "got " + ev.name
    if ev.name =~ /\.*/.minion$/
      torun << ev.name
      if queued < MAXQUEUE
        queue(torun.shift)
        queued += 1
      end
    elsif ev.name =~ /\.*/.out$/ && File.size(ev.name) > 0
      log "finished " + ev.name
      queued -= 1
      while (queued < MAXQUEUE) && (not torun.empty?)
        queue(torun.shift)
        queued += 1
      end
    end
  end
end

i.add_watch(".", Inotify::CLOSE_WRITE)

done = false
while not done
  sleep POLL
  if torun.empty? && queued == 0
    sleep POLL
    if torun.empty? && queued == 0
      done = true
    end
  end
end
t.terminate
```



# Performance

semigroups order 9

**1 CPU**  $\approx$  178 hours

# Performance

semigroups order 9

**1 CPU**  $\approx$  178 hours

**8 CPUs**  $\approx$  27 wall clock hours,  $\approx$  201 CPU hours

semigroups order 9

**1 CPU**  $\approx$  178 hours

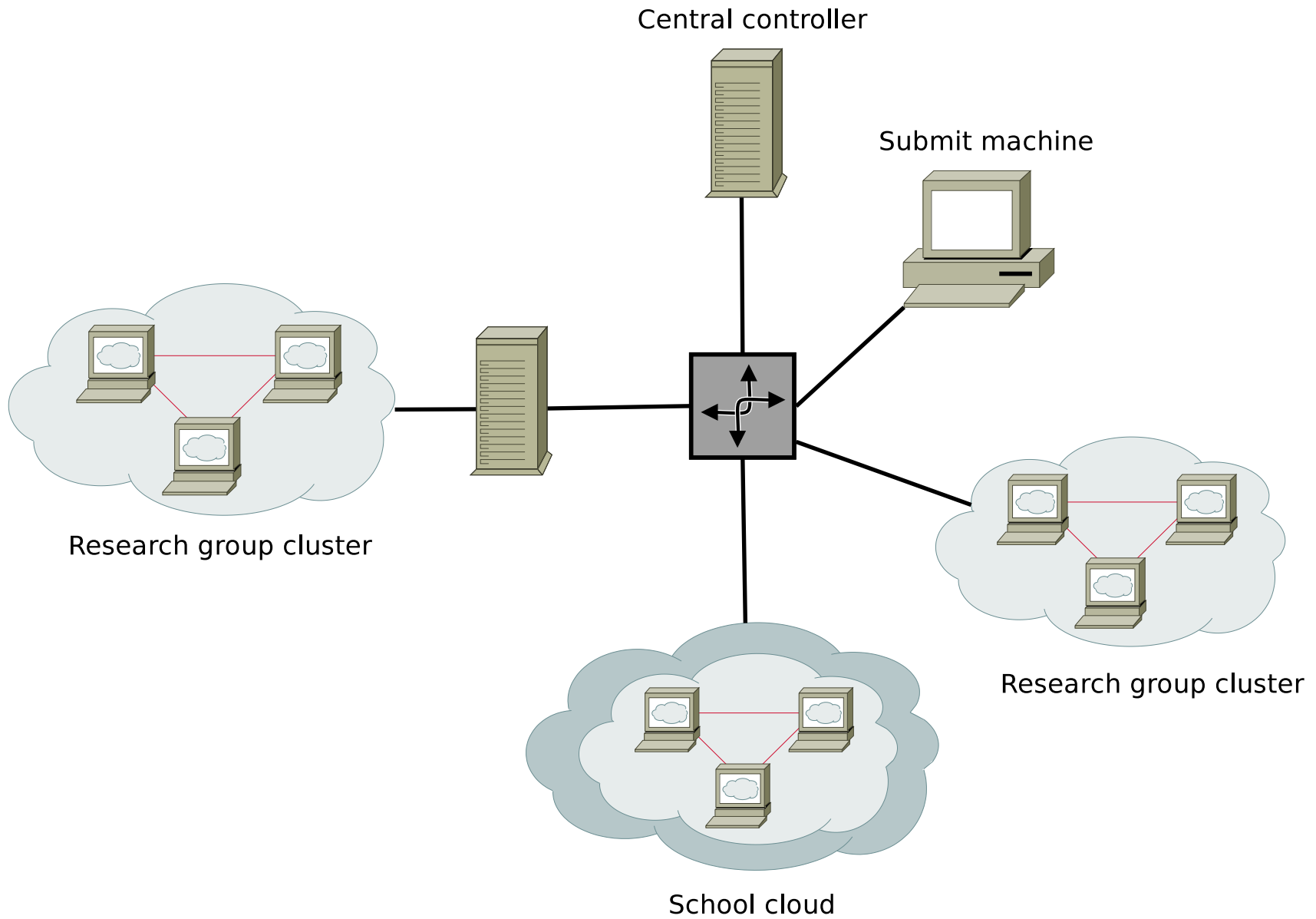
**8 CPUs**  $\approx$  27 wall clock hours,  $\approx$  201 CPU hours

**16 CPUs**  $\approx$  22 wall clock hours,  $\approx$  342 CPU hours

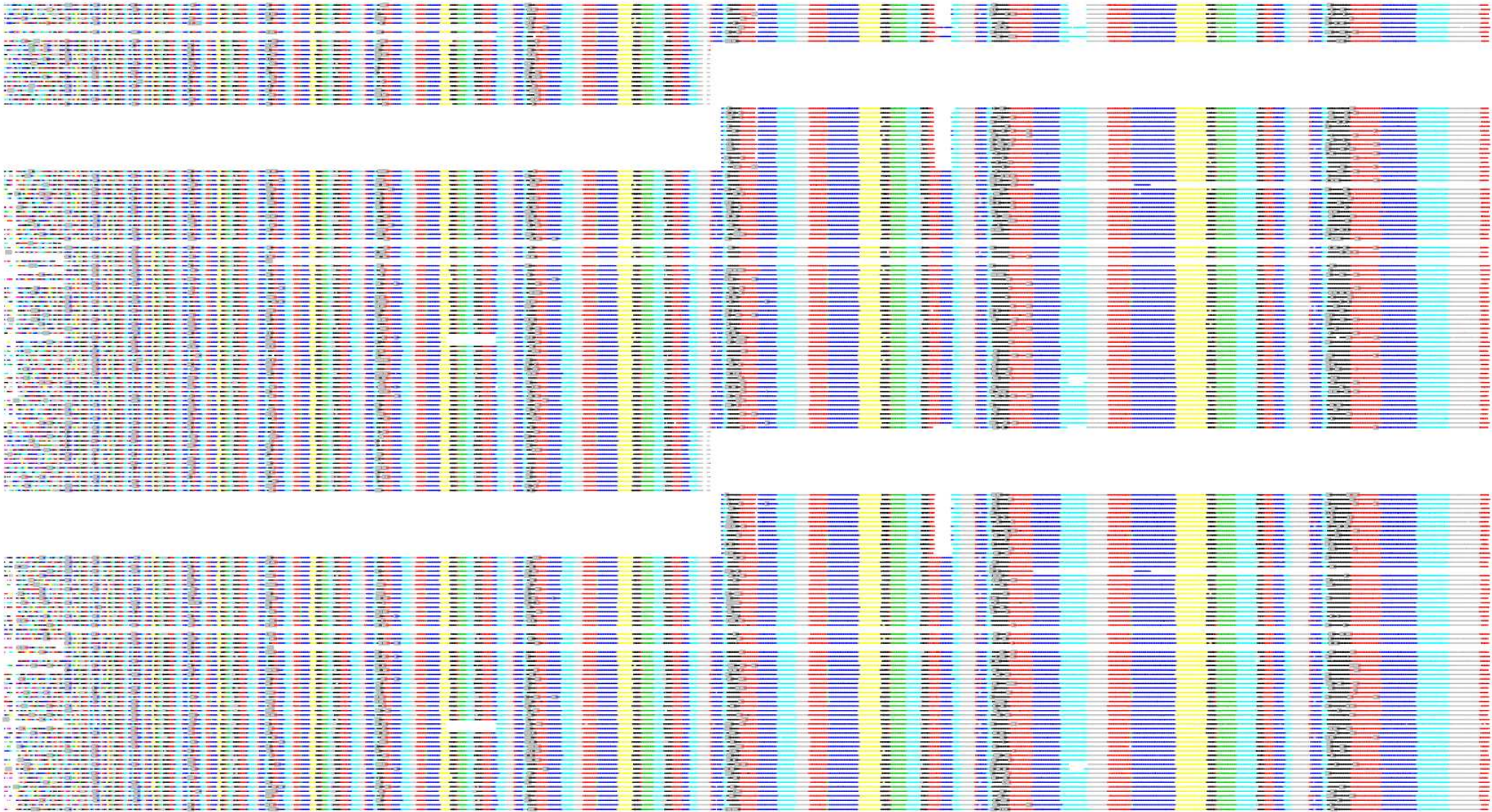
# Problem models

- ▷ 2830 Minion input files
- ▷ 2494 solve in  $< 1$  second
- ▷ 2631 solve in  $< 10$  seconds

# Used resources



# Job overview





Questions?

