The ClassAd Language

ClassAds: The *common language* in HTCondor

Classads: 3 uses

Description of entities in Condor

describes machines, jobs, services

Query language to select entities in Condor

“show me all the busy machines”

“show me idle jobs needing > 32 Gb ram”

2 way matching

Given jobs & machines, find matches

Classads *describe* all Entities

Entity	How to display full classad
Active Jobs	`$ condor_q -l`
Terminated Jobs	`$ condor_history -l`
Machines (slots)	`$ condor_status -l`
Finished jobs on machine	`$ condor_history -l -file $(condor_config_val STARTD_HISTORY)`
Active submitters	`$ condor_status -submitter -l`
Accounting records	`$ condor_userprio -l`
Schedd service	`$ condor_status -schedd -l`
All services	`$ condor_status -any -l`

Classads as *Job* Description

Set of *Attributes*

Attribute:

Key = Value

Key is a name

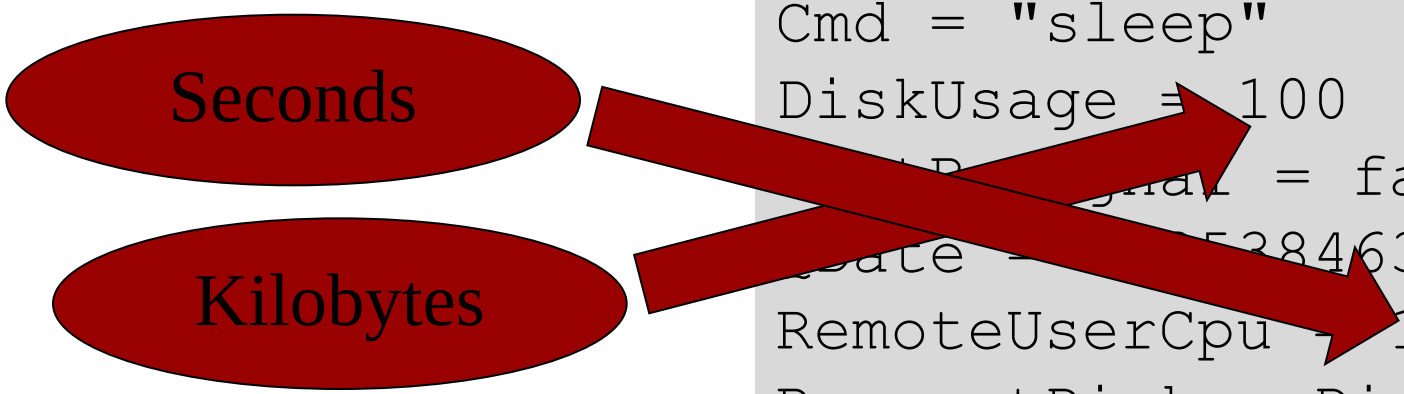
Value has a type

```
$ condor_q -l 180.0
```

```
ClusterId = 180
Cmd = "sleep"
DiskUsage = 100
ExitBySignal = false
QDate = 1535384632
RemoteUserCpu = 12.7
RequestDisk = DiskUsage
… (many attributes removed)
```

Classads as *Job* Description

Units by context




```
$ condor_q -l 180.0

ClusterId = 180
Cmd = "sleep"
DiskUsage = 100
[illegible] = false
[illegible]384632
RemoteUserCpu = 12.7
RequestDisk = DiskUsage
… (many attributes removed)
```

Manual lists all* attributes

http://htcondor.org (look for “Manual”)

Appendix A:

Lists all htcondor-defined attributes

And Units (if any) and how used

A.2 Job ClassAd Attributes

Absent: Boolean set to true `True` if the ad is absent.

AcctGroup: The accounting group name, as set in the submit description file via the **accounting_group** command. This attribute is only present if an accounting group was requested by the submission. See section 3.6.7 for more information about accounting groups.

AcctGroupUser: The user name associated with the accounting group. This attribute is only present if an accounting group was requested by the submission.

AllRemoteHosts: String containing a comma-separated list of all the remote machines running a parallel or mpi universe job.

Args: A string representing the command line arguments passed to the job, when those arguments are specified using the *old* syntax, as specified in section 12.

Arguments: A string representing the command line arguments passed to the job, when those arguments are specified using the *new* syntax, as specified in section 12.

BatchQueue: For grid universe jobs destined for PBS, LSF or SGE, the name of the queue in the remote batch system.

BlockReadKbytes: The integer number of KiB read from disk for this job.

BlockReads: The integer number of disk blocks read for this job.

BlockWriteKbytes: The integer number of KiB written to disk for this job.

BlockWrites: The integer number of blocks written to disk for this job.

BoincAuthenticatorFile: Used for grid type boinc jobs; a string taken from the definition of the submit description file command **boinc_authenticator_file**. Defines the path and file name of the file containing the authenticator string to use to authenticate to the BOINC service.

CkptArch: String describing the architecture of the machine this job executed on at the time it last produced a checkpoint. If the job has never produced a checkpoint, this attribute is `undefined`.

Attribute Names (before the =)

- Are like “C” (Python, R,Matlab…) identifiers
  - Must start with letter, then letters, numbers, _
  - No limit on length, but be reasonable
  - Case insensitive, but CamelCase is traditional
  - Extendable – you can add custom attributes
    - (covered in another talk)

Main ClassAd types

Type	Description
Boolean	*true, false*
Integers	*64 bit signed*
Reals	*64 bit IEEE 754 Double*
Strings	*"quoted"*
Reference	Lookup another attribute

Booleans

Booleans can be

- true
- false

Case-insensitive

- (True, TRUE)

Note – NO QUOTES

```
$ condor_q -l 180.0
```

```
ClusterId = 180
Cmd = "sleep"
DiskUsage = 100
ExitBySignal = false
QDate = 1535384632
RemoteUserCpu = 12.7
RequestDisk = DiskUsage
… (many attributes removed)
```

Classad Integers

64 bit

- Even on 32 bit binaries

Always signed

Overflow -> wrap quietly

No Hex prefix

- Don’t even ask about octal

1 / 0 -> error

```
$ condor_q -l 180.0
```

```
ClusterId = 180
Cmd = "sleep"
DiskUsage = 100
ExitBySignal = false
QDate = 1535384632
RemoteUserCpu = 12.7
RequestDisk = DiskUsage
… (many attributes removed)
```

Classad Reals

- IEEE 64 bit
  - And all the oddities
- Scientific Notation
  - -5.6e-5
- Overflow -> Infinity
- 1e990 -> real("INF")
- NaNs -> real("Nan")

```
$ condor_q -l 180.0
```

```
ClusterId = 180
Cmd = "sleep"
DiskUsage = 100
ExitBySignal = false
QDate = 1535384632
RemoteUserCpu = 12.7
RequestDisk = DiskUsage
… (many attributes removed)
```

Classad Strings

Must be quoted with "

Escape with backslash:

" foo\"bar“

No Other Escapes!

Hard to get newlines in string because of tools

```
$ condor_q -l 180.0
```

```
ClusterId = 180
Cmd = "sleep"
DiskUsage = 100
ExitBySignal = false
QDate = 1535384632
RemoteUserCpu = 12.7
RequestDisk = DiskUsage
… (many attributes removed)
```

Classad References

- Like variable lookup
- What is RequestDisk?
- Lookup DiskUsage,

- Return 100

```
$ condor_q -l 180.0
```

```
ClusterId = 180
Cmd = "sleep"
DiskUsage = 100
ExitBySignal = false
QDate = 1535384632
RemoteUserCpu = 12.7
RequestDisk = DiskUsage
… (many attributes removed)
```

Undefined

- Very Important to Grok
- Rarely explicit
  - ExitBySignal -> undefined
- MissingAttr -> undefined
- Means “Don’t Know”
- Could mean “missing”

```
$ condor_q -l 180.0
```

```
ClusterId = 180
Cmd = "sleep"
DiskUsage = 100
ExitBySignal = undefined
QDate = 1535384632
RemoteUserCpu = 12.7
RequestDisk = DiskUsage
… (many attributes removed)
```

More Undefined

- Allows decisions when information missing
- Context determines trueness or falseness:
- What does missing *ExitBySignal* mean?

Missing vs undefined

No difference!

```
$ condor_q -l 180.0

ClusterId = 180
Cmd = "sleep"
DiskUsage = 100

QDate = 1535384632
RemoteUserCpu = 12.7
RequestDisk = DiskUsage
… (many attributes removed)
```

More Undefined

- What does missing *ExitBySignal* mean?

Neither true nor false

Job hasn't exited (yet)?

Remote Site didn't tell us?

???

```
$ condor_q -l 180.0

ClusterId = 180
Cmd = "sleep"
DiskUsage = 100

QDate = 1535384632
RemoteUserCpu = 12.7
RequestDisk = DiskUsage
… (many attributes removed)
```

Classad Expressions

Expressions combine values

C/Java/Python-like:

Logical: *evaluate to boolean*

Math*: +, -, /, *, <<, >>, % evaluate to number*

Functions *(builtins) depends on function*

Logical Expressions

Expression	Meaning
>	Greater Than
<	Less Than
>=	Greater Than or equal
<=	Less Than or equal
&&	Logical And (short circuited)
||	Logical Or (short circuited)
==	Equality Test
!=	Inequality Test

Examples with Logicals

```
(cmd == “sleep”)
      -> true
```

```
$ condor_q -l 180.0
```

```
ClusterId = 180
Cmd = "sleep"
DiskUsage = 100
ExitBySignal = undefined
QDate = 1535384632
RemoteUserCpu = 12.7
RequestDisk = DiskUsage
… (many attributes removed)
```

Examples with Logicals

```
(DiskUsage > 100)
     -> false
```

```
$ condor_q -l 180.0
```

```
ClusterId = 180
Cmd = "sleep"
DiskUsage = 100
ExitBySignal = undefined
QDate = 1535384632
RemoteUserCpu = 12.7
RequestDisk = DiskUsage
… (many attributes removed)
```

Examples with Logicals

```
(RequestDisk >= 120)
      -> false
```

```
$ condor_q -l 180.0
```

```
ClusterId = 180
Cmd = "sleep"
DiskUsage = 100
ExitBySignal = undefined
QDate = 1535384632
RemoteUserCpu = 12.7
RequestDisk = DiskUsage
… (many attributes removed)
```

Examples with Logicals

```
ExitBySignal == true
    -> undefined
```

```
$ condor_q -l 180.0

ClusterId = 180
Cmd = "sleep"
DiskUsage = 100
ExitBySignal = undefined
QDate = 1535384632
RemoteUserCpu = 12.7
RequestDisk = DiskUsage
… (many attributes removed)
```

Examples with Logicals

```
(cmd == “sleep”) &&
(RemoteUserCpu > 1)
&&
((DiskUsage < 100))
      -> false
```

```
$ condor_q -l 180.0
```

```
ClusterId = 180
Cmd = "sleep"
DiskUsage = 100
ExitBySignal = undefined
QDate = 1535384632
RemoteUserCpu = 12.7
RequestDisk = DiskUsage
… (many attributes removed)
```

Math Expressions

Expression	Meaning
+	Addition
-	Subtraction (or unary minus)
/	Division
%	Modulus
*	Multiplication
>>	Bitwise shift right
<<	Bitwise shift left

Examples with Math

```
(DiskUsage * 1024)
     -> 102400
```

```
$ condor_q -l 180.0
```

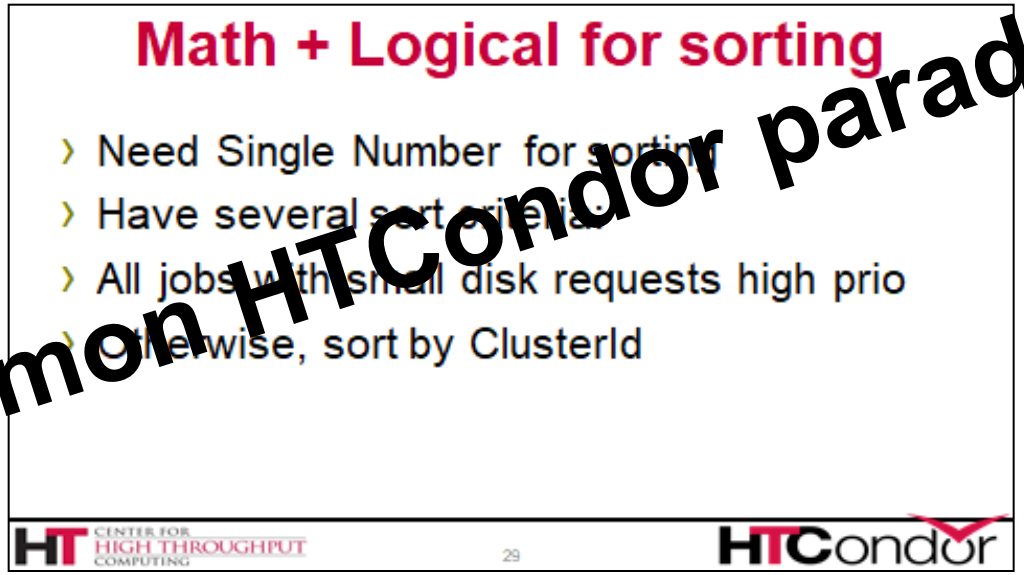
```
ClusterId = 180
Cmd = "sleep"
DiskUsage = 100
ExitBySignal = undefined
QDate = 1535384632
RemoteUserCpu = 12.7
RequestDisk = DiskUsage
… (many attributes removed)
```

Math + Logical for sorting

- Need Single Number for sorting
- Have several sort criteria:
- All jobs with small disk requests high prio
- Otherwise, sort by ClusterId

Booleans expand to int

((DiskUsage < 100) * 1000000) + ClusterId



Common HTCondor paradigm!

Bitwise Expressions

Expression	Meaning
|	Bitwise or
&	Bitwise and
^	Bitwise xor
>>	Bitwise shift right
<<	Bitwise shift left

Classad Builtin Functions

Expression	Returns
time()	Current time in seconds from epoch
substr(str, offset, len)	Extract substring
regexp(pattern, str)	Regexp match (pcre based)
random(x)	Random number from 0 to x
IsUndefined(expr)	True if expr is undefined
StringListMember(s, l)	Is s in list l, where l like “a, b, c”
toUpper(s)	Upper-case s

Examples with Functions

```
(QDate + 3600) > time()
      -> true (maybe)

Regexp("^s.*", Cmd)
      -> true

IsUndefined(foo)
      -> true
```

```
$ condor_q -l 180.0
```

```
ClusterId = 180
Cmd = "sleep"
DiskUsage = 100
ExitBySignal = undefined
QDate = 1535384632
RemoteUserCpu = 12.7
RequestDisk = DiskUsage
… (many attributes removed)
```

Eval/Unparse: Jedi Level

Requirements = WantGluster

Does WantGluster appear in Reqs?

Req’mts is *Expression*, not string

So regexp, etc. don’t work

Unparse: expr to string

Requirements = WantGluster

Unparse(Requirements) ->

“WantGluster”

```
regexp(“Gluster.*”, unparse(Req))
     -> True
```

Control Flow

- Expr ? tValue : fValue
  - If expr evals to True, return tValue, else fValue
- IfThenElse(expr, tValue, fValue)
  - ditto
- Expr ?: ReturnThisIfExprWasUndefined

My Favorite Function: Debug()

- Debug(anyExpression) -> anyExpression
- Thus Debug is a no-op
- Has a side effect:
  - DaemonLog traces expression evaluation

```
Requirements = WantGluster && (1024 > Memory)
```

```
Requirements = debug(WantGluster && (1024 > Memory))
```

Negotiator Log shows:

```
13:32:12 Classad debug: WantGluster --> UNDEFINED
13:32:12 Classad debug: 409600 --> 409600
13:32:12 Classad debug: [0.01001ms] Memory --> 409600
13:32:12 Classad debug: [0.03791ms] WantGluster &&
(1024 > Memory) --> FALSE
```

ClassAd Lists and Nesting

```
childCpus[1]
      -> 2

childCpus[SlotId]
      -> 4

Size(childCpus)
      -> 4
```

```
$ condor_status -l a_pslot
```

```
Name = "fastmachine"
ChildCpus = {1, 2, 3, 4}
slotId = 3


… (many attributes removed)
```

Nested ClassAd (not used (yet!))

```
childSlot.Name
      -> "slot1"

childSlot.Cpus
      -> 4

childSlot["name"]
      -> "slot1"
```

```
$ condor_status -l a_pslot
```

```
Name = "fastmachine"
ChildSlot =[
      Name = "slot1"
      Cpus = 4
]
Cpus = 40
… (many attributes removed)
```

Classads: On to 2nd use

Description of entities in Condor

describes machines, jobs, services

***Query language* to select entities in Condor**

"show me all the busy machines"

"show me idle jobs needing > 32 Gb ram"

2 way matching

Given jobs & machines, find matches

Query language

- Users can write *expressions* as *queries*
- These *select* a subset from a larger set
- If condor evaluates expression to *TRUE*

Query Language example

$ condor_status –const "some classad expr"

$ condor_q –const "some classad expr"

Example query

```
$ condor_status -const "State == \"Busy\""
$ condor_status -const "State != \"Busy\""
```

```
MachineName = "Machine1"
State = "Busy"
MemoryUsage 1024
***
MachineName = "Machine2"
State = "Idle"
***
MachineName = "Machine3"
State = "Busy"
MemoryUsed = 2048
```

Example query

```
$ condor_status –const “MemoryUsage > 2000”
```

```
MachineName = “Machine1”
State = “Busy”
MemoryUsage 1024
***
MachineName = “Machine2”
State = “Idle”
***
MachineName = “Machine3”
State = “Busy”
MemoryUsed = 2048
```

Squashing operators

- What does the expression

  “Some String” == undefined

  Or

  “Some String” == reference to missing value

  Evaluate to?

How About

“Some String” != undefined

Or

“Some String” != reference to missing value

Evaluate to?

- “foo” == undefined -> undefined
- “foo” != undefined -> undefined

- Sometimes you want
- “foo” != undefined to mean false.

Enter the Squashing operators

- =?= and =!= are *Squashing* comparisons

- And NEVER return undefined:
- “Some String” =?= undefined -> false
- “Some String” =!= undefined -> true

Example Squash

```
$ condor_status -const "State != \"Busy\""
$ condor_status -const "State =!= \"Busy\""
```

```
MachineName = "Machine1"
State = "Busy"
MemoryUsage 1024
***
MachineName = "Machine2"
***
MachineName = "Machine3"
State = "Busy"
MemoryUsed = 2048
```

More Squashing

- String == is case **IN**sensitive
- String =?=, =!= is case sensitive (!)
- Trivia:
- Undefined == undefined ->
- Undefined =?= undefined ->

Classads: 3rd use

Description of entities in Condor

describes machines, jobs, services

Query language to select entities in Condor

“show me all the busy machines”

“show me idle jobs needing > 32 Gb ram”

2 way matching

Given jobs & machines, find matches

Matchmaking

Requires *TWO ads*, returns *true* or *false*

"In the context of ad1 and ad2"

With a selection expression in the

Requirements value of both ads

Commonly used to match jobs and machines

References when matching

Reference: lookup

e.g. OldName -> “Foo”

```
IsGood = true
IsNotGood = false
RunTime = 123
Name = “Foo”
OldName = Name
Price = 23.45
Foo = undefined
U = Missing
```

References when matching

```
IsGood = true
RunTime = 123
Name = "Foo"
OldName = Name
Price = 23.45
Foo = undefined
U = Missing
```

```
IsGood = true
RunTime = 123
Name = "Bar"
Price = 23.45
Foo = undefined
U = Missing
```

What does OldName return now?

References when matching

- Ads are ordered, “my” and “target”
- Lookup first in my, then target

References with My and Target

- Prefix reference with “MY.” or “Target.”
- To force lookup in one side or the other
- Rarely used, but good idea

References when matching

```
IsGood = true
RunTime = 123
Name = “Foo”
OldName = TARGET.Name
Price = 23.45
Foo = undefined
U = Missing
```

```
IsGood = true
RunTime = 123
Name = “Bar”
Price = 23.45
Foo = undefined
U = Missing
```

What does OldName return now?

References when matching

```
IsGood = true
RunTime = 123
Name = “Foo”
OldName = TARGET.Name
Price = 23.45
Foo = undefined
U = Missing
```

```
IsGood = true
RunTime = 123
Name = “Bar”
Price = 23.45
Foo = undefined
U = Missing
```

What does TARGET.Name return?

References when matching

```
IsGood = true
RunTime = 123
OldName = Name
Price = 23.45
Foo = undefined
U = Missing
```

```
IsGood = true
RunTime = 123
Name = “Bar”
OldName = Name
Price = 23.45
Foo = undefined
U = Missing
```

What does OldName return now?

For 2 ads to match, both Requirements -> true

- Evaluate Requirements of one, if true
- Evaluate Requirements of other.
- Note My and Target swap

Job Ad

```
Type   = "Job"
Requirements =
    HasMatlabLicense
      =?= True
Cmd= "/bin/sleep"
Args = "3600"
Owner = "gthain"
NumJobStarts = 8
```

Slot Ad

```
Type = "Machine"
Cpus   = 40
Memory = 2048
Requirements =
  (Owner == "gthain")   &&
  (TARGET.NumbJobStarts <=
   MY.MaxTries)
HasMatlabLicense = true
MaxTries = 4
```

Advanced Topics

- User-defined classad functions
- Python bindins
- Standalone classads

Questions?

Thank You!