Importance of Data Locality

- Gerald

This is summary of my research on how data locality affects performance.
Detailed doc is here https://docs.google.com/document/d/17GYvQZrChH_3YC_F8_xpNTYldIfqvu4MUmHJOZPY_78/edit?hl=en#

Abstract

For data parallel systems (Hadoop, Sector/Sphere, etc), computation nodes and storage nodes are not separated. In other words, usually a node has the responsibility of both storage and computation. For this architecture, two natural problems are how to place data and how to schedule computation jobs. One heuristic is that the systems should put data and computation as close as possible. This is the scheduling policy used by Hadoop. Job scheduler schedules a map task to the node where input data is stored. **Our goal is to evaluate importance of data locality in data parallel systems. In other words, we will study tradeoffs between migration of computing versus migration of data.**
In this document, we designed a set of tests to figure out importance of data locality in modern data parallel systems. **We focus on I/O intensive applications**.

Scheduling Policies

- **Default**: default scheduler provided by Hadoop
  Hadoop scheduler tries to schedule a task to a node where input data is stored. However, if all such nodes are saturated (number of running tasks is equal to maximum number of tasks on each node) , the scheduler dispatches the task to an idle node picked randomly..
- **Random scheduling**:
  schedule tasks to randomly selected non-saturated nodes.
- **Customized random**:
  Choose between data-local scheduling and random scheduling according to a parameter - *randomness*. It represents the probability that random scheduling is chosen. If its value is 0, scheduling becomes data-local scheduling. If its value is 1, scheduling becomes pure random scheduling.

Test Applications

We have two test programs which are heavily I/O intensive.

- Line count
  Count number of lines in input files.
- Input sink
  Read input records from input files and throw them away. Don't do any computation. No reduce task.

Evaluation metrics

There are three main **parameters** which control the test environment

1. **size of input**
2. **number of nodes**
3. **max number of maps per node**

Main **measurements** include

1. Job run time
   How long does each job run
   a. Job run time is positively correlated to size of input.
      For map-only applications, job run time is directly proportional to input size if we assume that map operations applied to records task approximate amount of time. For jobs with reduce phase, job run time depends on 1) data shuffling between map tasks and reduce tasks, and 2) influence of size of reduce input on reduce phase run time.
   b. Job run time is negatively correlated to number of nodes to some point.
      Because of overhead of using and managing large number of tasks, adding more

nodes to increase concurrency may not be always beneficial. Especially, when a task runs short (e.g. 3 seconds), overhead of starting up and tearing down the task may outweigh task processing time itself.

c. Job run time is negatively correlated to max number of maps per node when parallelization benefit outweighs resource competition. Otherwise, it is proportional. The transition point is crucial.

2. Average map task run time
   How long does a map task run on average?

Denotation:

N: number of nodes
D: size of input
T: job run time
M_{max}: max number of maps per node

Normalization

Given a fixed amount of data to process, ideally the speedup should be

$M_{max} * N$

compared with the baseline configuration where M_{max} and *N* are both 1.

Absolute job run time cannot reveal efficiency of scaling.

- When number of nodes is increased, performance is improved most of the time. But efficiency of scaling-out may be low. For example, it's possible that performance is improved by 2 times while number of nodes is increased by 100 times.
  So to erase effect of number of nodes, absolute job run time should be normalized.

  Fix *M*, assume *I* is the time needed to process one unit of data using one node, we have

  $T = I * D / N \Rightarrow I = T * N / D$. (T, N, D are known from test result)

- The same holds for *M*.
  We know that $I = T * N / D$ given a *M*. Now we take *M* into consideration.
  Let I_2 denotes the time needed to process one unit of data using one node by one map task. We assume all map slots are used during processing.

  $I = I_2 / M_{max} \Rightarrow I_2 = I * M_{max} = T * N * M_{max} / D$

- *I* and I_2 from above two bullets are normalized metrics.

Tests in Hadoop

Test environment

FutureGrid clusters.

During the test, **only one job runs at any moment in Hadoop cluster**. But because the job uses all available map slots, test results also apply to situation where multiple jobs run simultaneously.

Tests

I have done tests with following configurations:

41 nodes, 400G input data
31 nodes - 300G input data
31 nodes - 400G input data
21nodes - 200G input data
21 nodes - 400G input data
10 nodes - 400G input data
10 nodes - 100G input data
10 nodes - 100G input data, 64K buffer size
10 nodes - 100G input data, 1m buffer size
41 nodes, 400G inpput data, 192MB mapreduce split size, 64MB block size
41 nodes, 400G inpput data, 128MB mapreduce split size, 64MB block size
41 nodes, 400G inpput data, 256MB mapreduce split size, 64MB block size

Test results are put into following spreadsheet:
https://spreadsheets.google.com/ccc?key=0AnYKsmvRoSiodFFCaXRfQUcxbC1vX3ZKWVpZSnJENkE&hl=en&authkey=CI3A0JgK
Because test programs **inputsink** and **linecount** gives similar results,only test results of inputsink is included in the spreasheet.

Plots and details of tests are written in doc https://docs.google.com/document/d/17GYvQZrChH_3YC_F8_xpNTYldIfqvu4MUmHJOZPY_78/edit?hl=en#. Here I just put observations below.

Observations

Job run time vs. M_{max}

- The relationship is NOT linear. Ideally, job run time should be **inversely proportional** to M_{max}. But the plots show this is not the case.
  When M_{max} is small, increasing M_{max} can bring significant benefit so that job run time is decreased drastically. This works because increasing M_{max} also increases concurrency.
  When M_{max} becomes big (more than 32), job run time does not change much. For some tests, to increase M_{max} from 32 to 64 even results in increase of job run time. This means competition of resources outweighs benefit of higher concurrency.
- Random scheduling performs worse than default scheduling.
- There is additional overhead when 1) number of maps is increased 2) input size is increased.
    - The more maps run on each node, the more overhead it incurs.

- The larger input data is, the more overhead it incurs.

Following table shows job run time of tests run on **20 nodes**.

0.5 random				default				random			
Maps	**400G**	**100G**	**Times**	**Maps**	**400G**	**100G**	**Times**	**Maps**	**400G**	**100G**	**Times**
1	5739	1327	4.32	1	5711	1329	4.29	1	5754	1321	4.35
2	2686	582	4.61	4	2554	538	4.74	4	2761	625	4.41
4	1822	404	4.50	8	1868	315	5.93	8	1932	438	4.41
8	1643	360	4.56	10	1594	277	5.75	10	1869	370	5.05
10	1193	248	4.81	16	1187	231	5.13	16	1231	262	4.69
32	1018	207	4.91	32	988	151	6.54	32	1107	185	5.98
64	1040	111	9.36	64	955	138	6.92	64	1050	168	6.25

Field “Times” is result of dividing run time with 400G input data by run time with 100G input data.

- From above table, we can see the more data is processed, the longer it takes.
- Job run time is not inversely proportional to input data size.

  slowdown = $\frac{job\,run\,time\,of\,large\,data}{job\,run\,time\,of\,small\,data}$

  As M_{max} is increased, slowdown is increased even through job run time is decreased. This means when M_{max} is big, processing large amount of data is not efficient compared with processing small amount of data.

Job run time vs. number of nodes

Following table shows speedup resulting from increase of number of nodes. (400GB data is processed)
For Hadoop cluster, one node is master and other nodes are slave.

		(num of slave nodes)				(num of slave nodes)			
	maps	**40**	**30**	**20**	**10**	**40**	**30**	**20**	**10**
default	1	1431	1904	2852	5711	3.99	2.99	2.00	1.00
	4	608	787	1180	2554	4.20	3.24	2.16	1.00
	8	422	463	756	1868	4.42	4.03	**2.47**	1.00
	10	331	434	699	1594	**4.81**	3.67	2.28	1.00
	16	261	354	572	1187	4.54	3.35	2.07	1.00
	32	222	259	440	988	4.45	**3.81**	2.24	1.00
	64	296	294	443	955	3.22	3.24	2.15	1.00
						4.23	3.47	2.19	
		(above is job run time)				(Above is speedup)			

0.5 random	1	1460	1910	2856	5739	3.93	3.00	2.00	1.00
	4	667	880	1298	2686	4.02	3.05	2.06	1.00
	8	481	612	931	1822	3.78	2.97	1.95	1.00
	10	399	530	787	1643	4.11	3.10	2.08	1.00
	16	305	382	548	1193	3.91	3.12	2.17	1.00
	32	226	254	394	1018	**4.50**	**4.00**	**2.58**	1.00
	64	281	288	408	1040	3.70	3.61	2.54	1.00
		(job run time)				3.99	3.26	2.19	
random	1	1459	1905	2853	5754	3.94	3.02	2.01	1.00
	4	706	957	1370	2761	3.91	2.88	2.01	1.00
	8	506	645	1026	1932	3.81	2.99	1.88	1.00
	10	440	557	836	1869	4.24	3.35	2.23	1.00
	16	328	418	634	1231	3.75	2.94	1.94	1.00
	32	245	271	461	1107	**4.51**	**4.08**	**2.40**	1.00
	64	299	278	534	1050	3.51	3.77	1.96	1.00
		(job run time)				3.95	3.29	2.06	

Following table shows speedup **all referred to the case M_{max}=1, N_{slave_nodes} = 10 and default scheduling**.

		(num of slave nodes)				(num of slave nodes)			
	maps	**40**	**30**	**20**	**10**	**40**	**30**	**20**	**10**
default	1	1431	1904	2852	5711	4.0	3.0	2.0	1.0
	4	608	787	1180	2554	9.4	7.2	4.8	2.2
	8	422	463	756	1868	13.5	12.3	7.6	3.0
	10	331	434	699	1594	17.2	13.1	8.2	3.6
	16	261	354	572	1187	21.9	16.1	10.0	4.8
	32	222	259	440	988	25.7	22.0	13.0	5.8
	64	296	294	443	955	19.3	19.4	12.9	6.0
		(above is job run time)				(Above is speedup)			
0.5 random	1	1460	1910	2856	5739	3.9	3.0	2.0	1.0
	4	667	880	1298	2686	8.6	6.5	4.4	2.1
	8	481	612	931	1822	11.9	9.3	6.1	3.1
	10	399	530	787	1643	14.3	10.8	7.2	3.5
	16	305	382	548	1193	18.7	14.9	10.4	4.8
	32	226	254	394	1018	25.3	22.5	14.5	5.6
	64	281	288	408	1040	20.3	19.8	14.0	5.5
		(job run time)				(Above is speedup)			

random	1	1459	1905	2853	5754	3.9	3.0	2.0	1.0
	4	706	957	1370	2761	8.1	6.0	4.2	2.1
	8	506	645	1026	1932	11.3	8.9	5.6	3.0
	10	440	557	836	1869	13.0	10.2	6.8	3.0
	16	328	418	634	1231	17.4	13.7	9.0	4.6
	32	245	271	461	1107	23.3	21.1	12.4	5.2
	64	299	278	534	1050	19.1	20.5	10.7	5.4
		(job run time)				(Above is speedup)			

- As we expect, the job completion time is decreased as we increase number of nodes.
- Default scheduling has highest speedup. Random scheduling has low speedup.
- The speedup is proportional to number of nodes.
- Speedup gains its maximum value when number of maps is not too small or large. Initially, as number of maps increases, speedup increases as well.At some point, speedup becomes maximum. Beyond that point, speedup decreases as number of maps is increased.

Normalized job run time *I* (eliminate factor of number of nodes)

1. When M_{max} is small (1 - 4)
    a. size of input data does not have significant impact on *I*.
2. When M_{max} becomes big,
    a. the more data is processed, the bigger *I* becomes.
3. When M_{max} is small, no matter how many nodes the system has and how much data is processed, *I* is pretty close.
   As M_{max} is increased, the difference between *I* with different input data size is increased.

Normalized job runme I_2 (eliminate factors of number of nodes and M_{max})

- Efficiency gets worse as M_{max} is increased.
  This means increasing concurrency by increasing M_{max} always results in decrease of efficiency.
  In worst case, it is 10x slower than ideal speedup.
- When M_{max} is small, no matter how many nodes the system has and how much data is processed, *I* is pretty close.
  As M_{max} gets larger (beyond 16), different system configurations in terms of number of nodes and amount of input data result in diverse I_2.

Map execution time

- For fixed number of nodes, the larger the input data is, the longer each map task executes.

- Random scheduling performs worse than default scheduling. The more nodes there are in the system, the smaller is the performance difference.
- As M_{max} is increased, map execution time is increased as well.
- Combining above three observations, we get the conclusion that the average map task run time is proportional to

- Both mean and standard deviation increases monotonically with increase of M_{max}. However, the slope changes.
  So one subsequence of increasing M_{max} is increase of map execution time. This is because of competition of resource usage.
- So increasing M_{max} is a double-edged sword.
  a. It increases concurrency by P times.
  b. It increases overhead by Q times because of resource usage contention.

  The final outcome depends on effect of above two factors. If $P > Q$, it decreases job run time. If $P < Q$, it increases job run time.
  P/Q = speedup => Q=P/speedup

Importance of Data locality

When Hadoop runs in cluster with high speed connections, whether data locality is critical depends on M_{max}.

- I assume nodes in a FutureGrid cluster are located on 1 rack.
  Based on my test, the speed of switch is 1Gbps. So pairwise connection among nodes is 1Gbps. This is comparable to disk I/O throughput.

- Random scheduling performs worst.

- Approximately, when M_{max} changes from 1 to 8, the slowdown of random scheduling (compared with default scheduling) is between **0% - 50%** and proportional to M_{max}. This means **data locality is increasing critical** with increase of M_{max}.
  When M_{max} becomes larger than 8, slowdown drops. This means **data locality becomes less critical**.
  When M_{max} is larger than 32, data locality is **not critical at all**. One explanation is that resource competition and other overhead outweigh effect of data locality. For some tests, random scheduling performs even better than default scheduling.

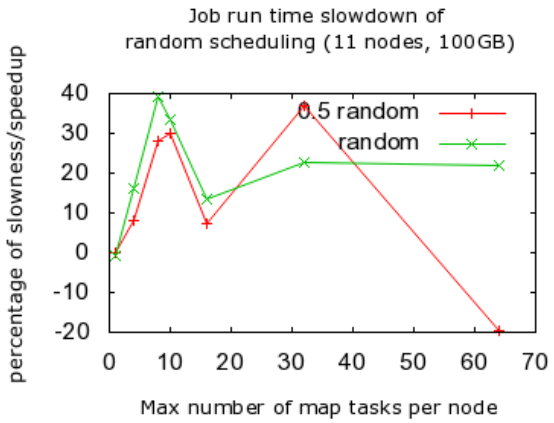
- For Hadoop cluster on Wide Area Network, we guess data locality is more critical generally.
-

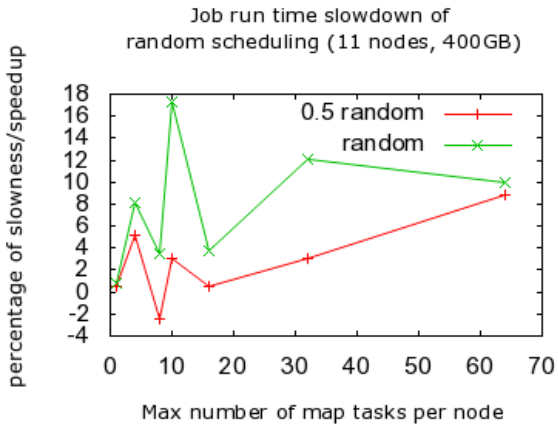
Single Cluster

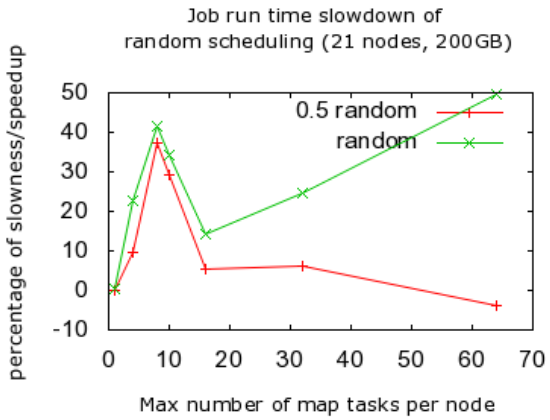
Following plots show **slowdown of random scheduling**:

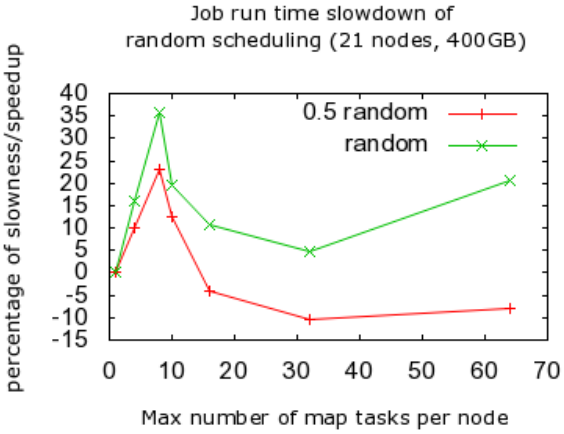
(job_runtime_random - job_runtime_default) / job_runtime_default

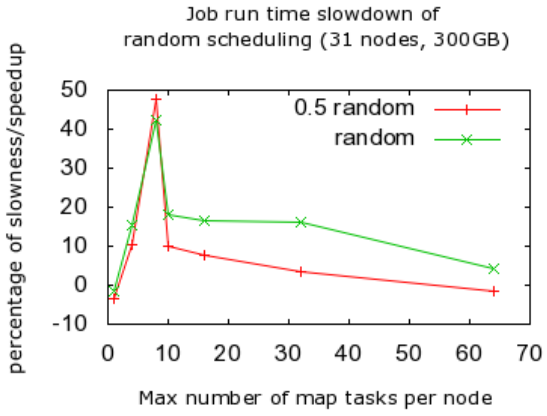
So the baseline is default scheduling.

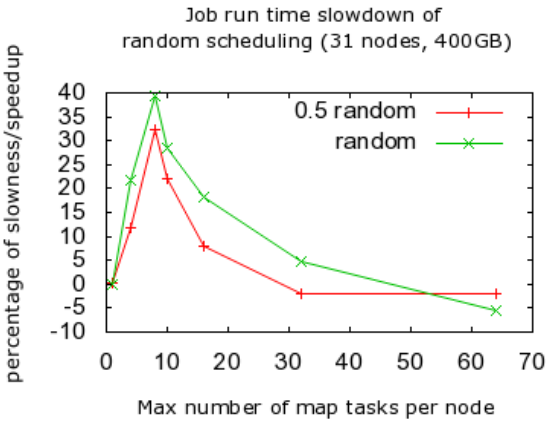


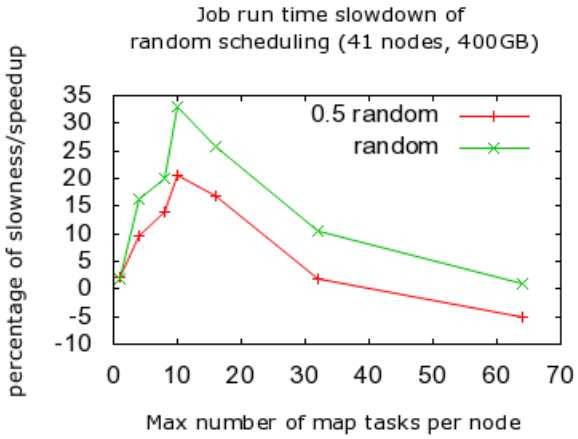






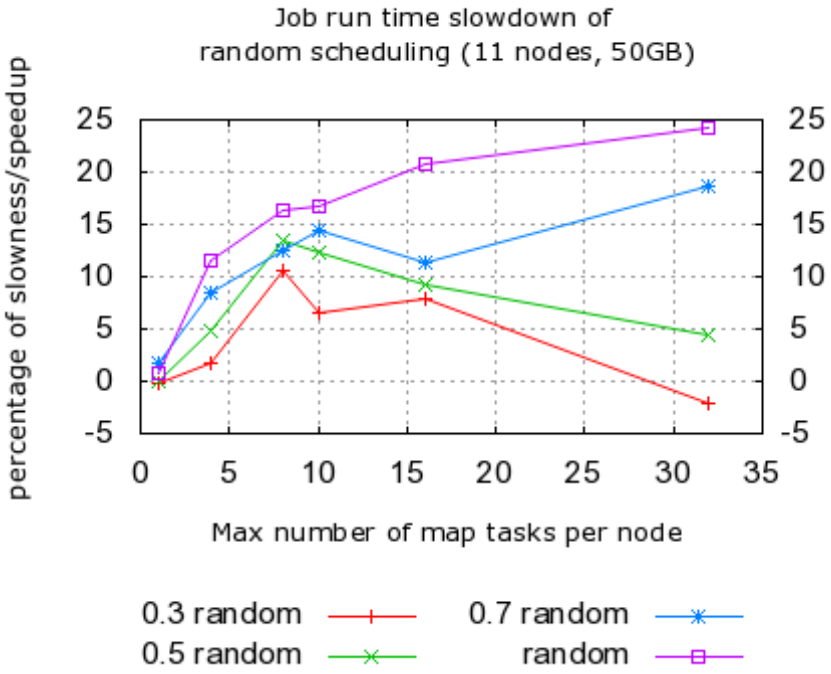






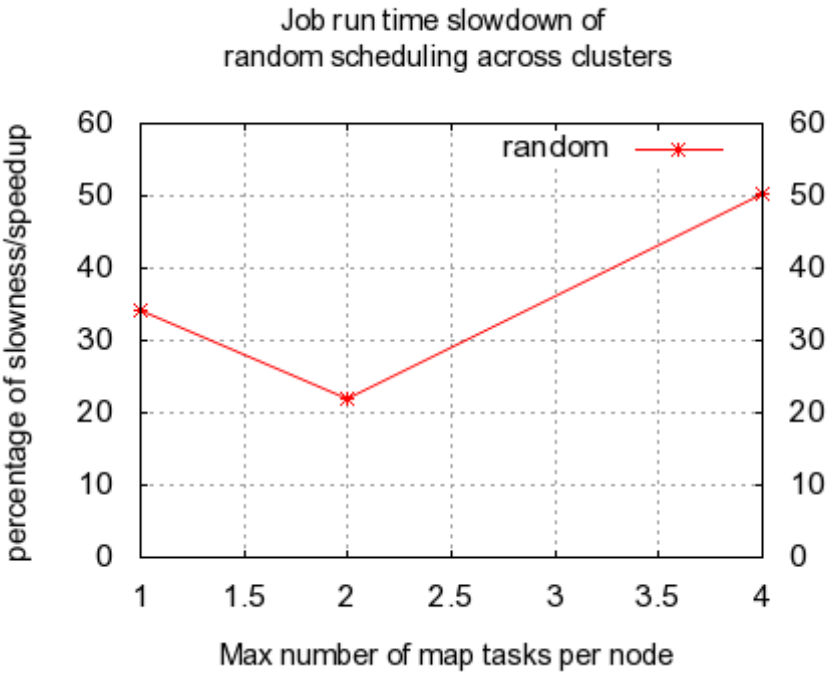
Randomness

Following tests are done in Alamo using 11 nodes (1 master and 10 slave) to process 50GB data. Randomness is varied (0.3 random, 0.5 random, 0.5 random, random) to see its effect on performance. Again the baseline is default scheduling.



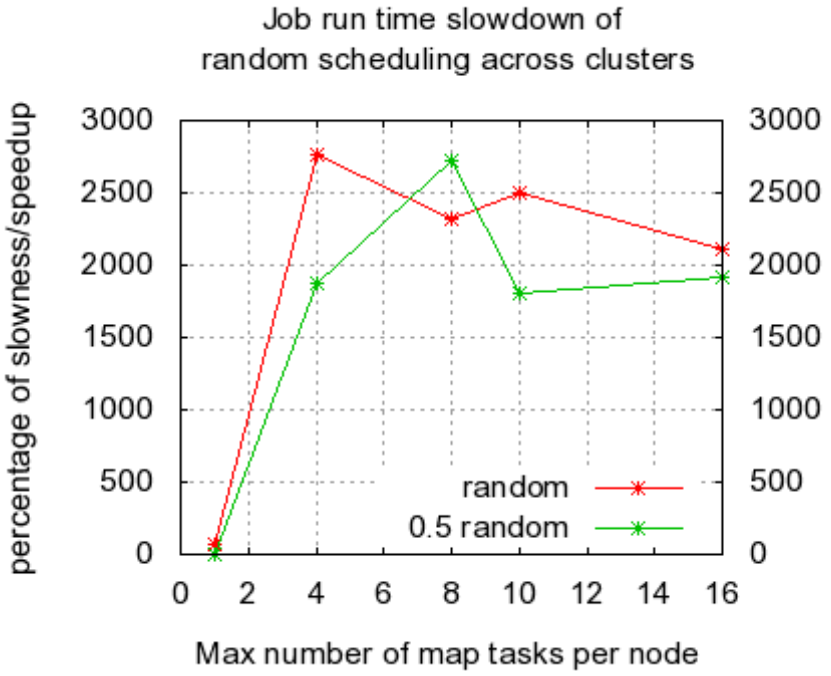
Cross cluster

1 EC2 instance and 1 polargrid machine are used. M_{max} is 1, 2 and 4.



Foxtrot and Sierra, cross-cluster tests (HDFS and MapReduce use the same nodes)

5 VM instances from foxtrot and 5 VM instances from sierra, ViNe is used to build virtual network. 1GB input data. Because of abnormally slow interconnection (compared with maximum possible throughput), this test result is not accurate.



From the plot, we see that random scheduling degrades performance a lot (around 2500%). When M_{max} is 1, default scheduling and random scheduling both have non-local scheduling (remember that default scheduling uses **best-effort** strategy to satisfy data-locality requirement). Because of slow cross-cluster interconnection (described in detail below), one non-local map task can slow down the whole process a lot. Rack-local data access is faster than cross-cluster data access by 510 times.

The reason is slow interconnection between VM instances in foxtrot and VM instances in sierra.

Pair of nodes tested	Network throughput	Tools used
foxtrot vm1 ↔ foxtrot vm2 (TCP)	941 Mbps	netperf
sierra vm1 ↔ sierra vm2 (TCP)	940 Mbps	netperf
foxtrot head node ↔ sierra head node (TCP)	60 Mbps	scp
foxtrot vm ↔ sierra vm (TCP)	1.84 Mbps	netperf (use -l 200 based on Mauricio's comment)
foxtrot vm ↔ sierra vm (UDP)	280 Mbps	netperf (with -l 200)

Table 3. Network throughput test

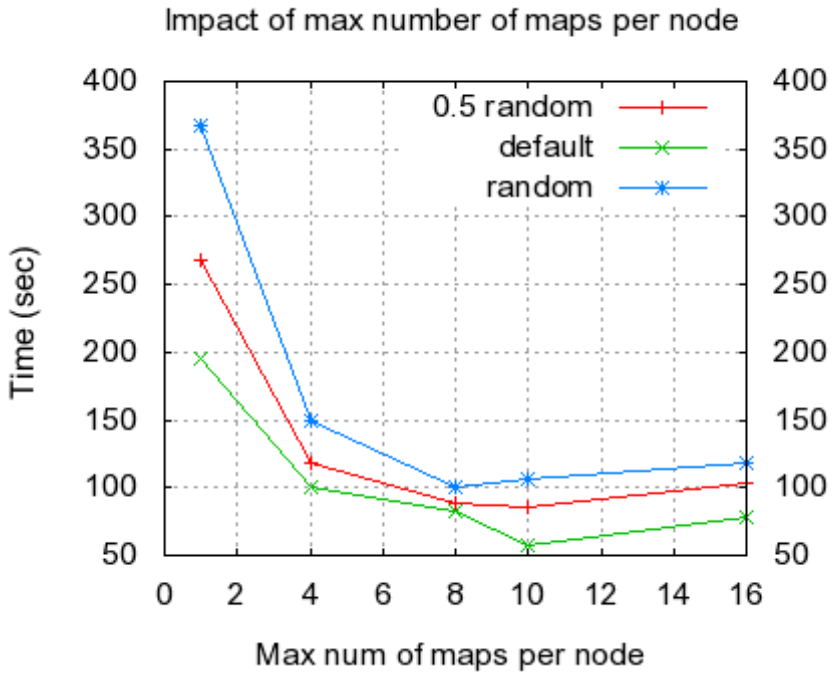
Several observations

1. VM layer does not incur much overhead because inter-VM connection **with the same cluster** is close to theoretical limit (I assume it is Giga network)
2. Interconnection between foxtrot and sierra is much lower than Giga speed.
3. On virtual network, cross-cluster connection becomes slower compared with physical network without ViNe being involved.
   Mauricio said that ViNe has been tested to process traffic up to 800Mbps.

For item 2 and 3, still not clear about what is the cause.

Sierra and Hotel, cross-cluster tests (HDFS and MapReduce use the same nodes)

Test environment: 5 Nimbus VM's in Hotel, 5 Nimbus VM's in Sierra. 10GB input data.
Note: **ViNe is not used**.



Throughput of network connection is shown below:

Pair of nodes tested	Network throughput	Tools used
hotel vm ← sierra vm (TCP)	268.52 Mbps	netperf, runs for 20 seconds
hotel vm → sierra vm (TCP)	107.48 Mbps	netperf, runs for 20 seconds
hotel vm ← sierra vm (TCP)	133.83 Mbps	netperf, runs for 100 seconds
hotel vm → sierra vm (TCP)	91 Mbps	netperf, runs for 100 seconds
hotel vm ← sierra vm (TCP)	54 Mbps	netperf, runs for 200 seconds
hotel vm → sierra vm (TCP)	69.03 Mbps	netperf, runs for 200 seconds
hotel vm ← sierra vm (TCP)	48 Mbps	netperf, runs for 300 seconds

hotel vm → sierra vm (TCP)	126.68 Mbps	netperf, runs for 300 seconds
hotel vm ← sierra vm (UDP)	903.10 Mbps	netperf, runs for 200 seconds
hotel vm → sierra vm (UDP)	961.52 Mbps	netperf, runs for 200 seconds

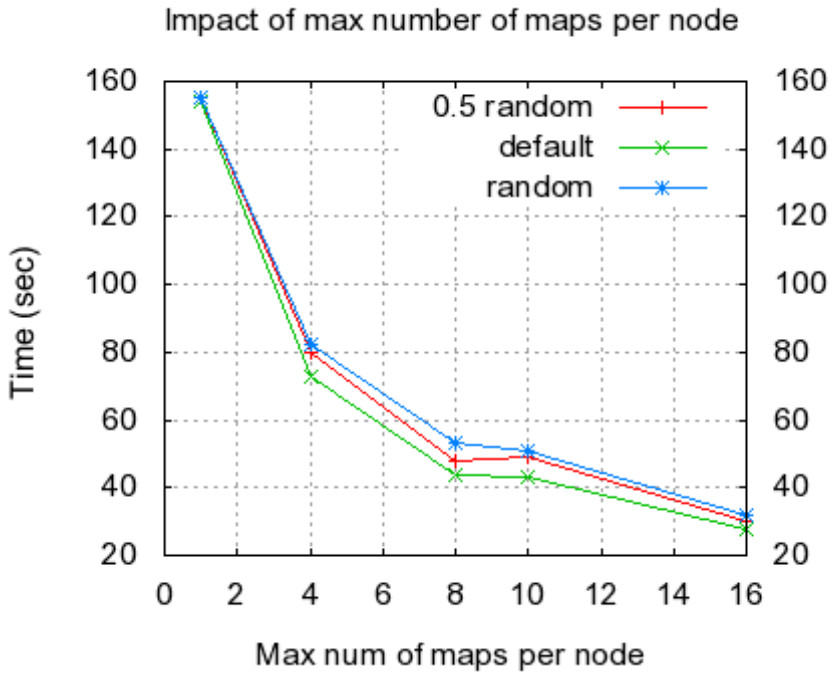
India and Hotel, cross-cluster tests (HDFS and MapReduce use the same nodes)

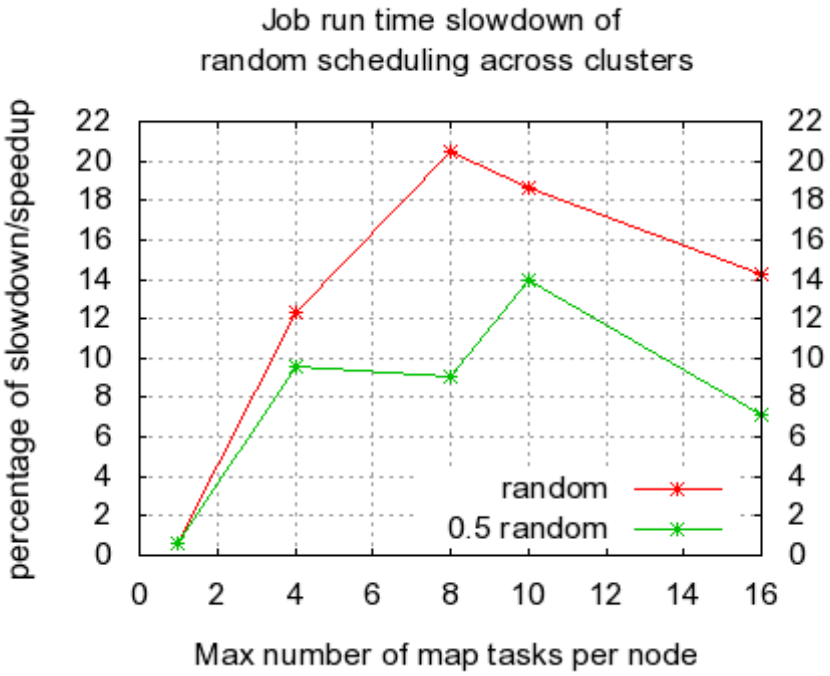
5 physical nodes from hotel and 5 physical nodes from india. 10GB input data.

Following plots show impact of random scheduling. First plot shows job run time and second plot shows percentage of slowdown.

From the plots, we can tell job run time is negatively correlated to randomness. The more random the scheduling policy is, the longer a job runs.

Approximately, random scheduling (100% random) slows down job execution by 2% - 20%.



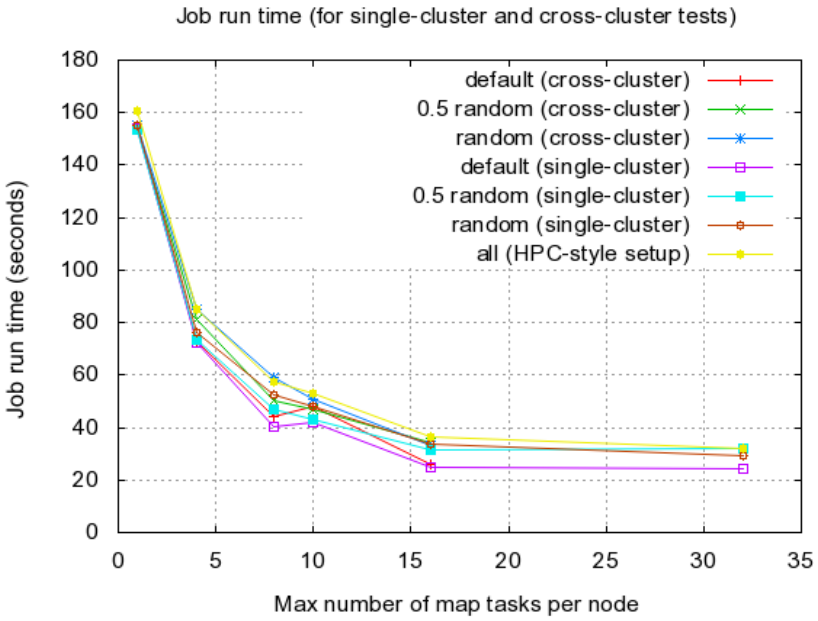


Comparison with single-cluster tests

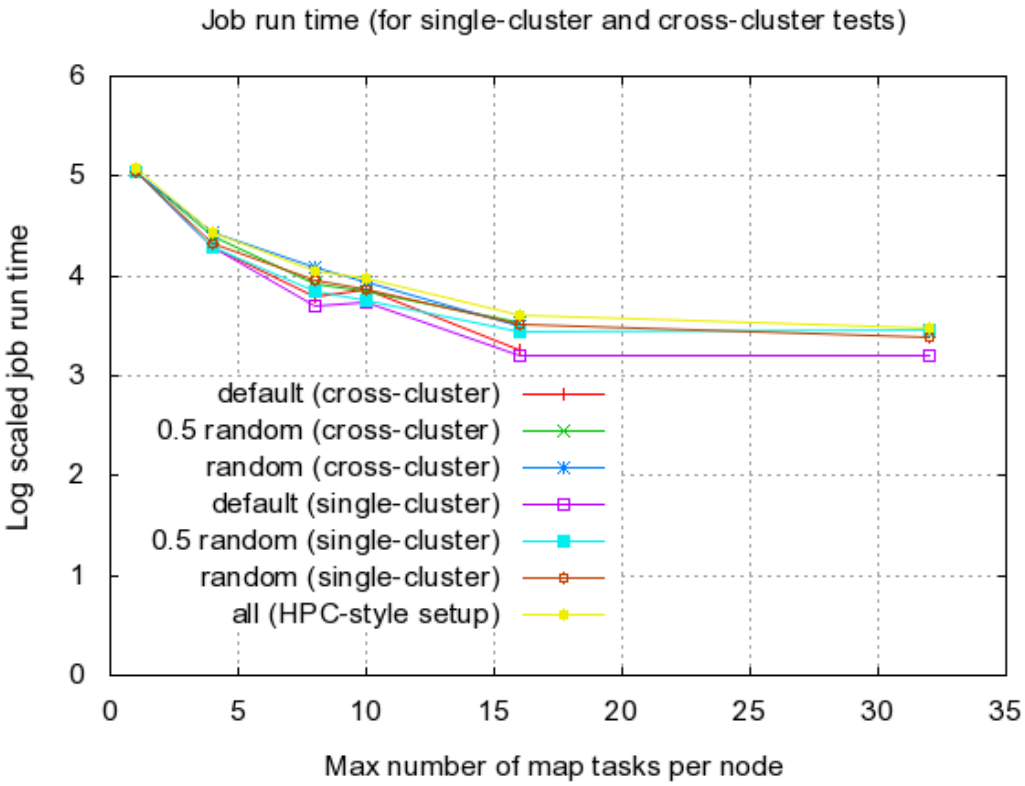
We ran both cross-cluster and single-cluster tests.

Cross-cluster tests: 5 nodes in India and 5 nodes in Hotel
Single-cluster tests: 10 nodes in India
Data size: 10GB
Following three plots show the comparison. Note: they show the same results in different ways.

- **Raw job run time comparison:**



- **Log-scaled run time comparison (logarithm is applied to run time):**

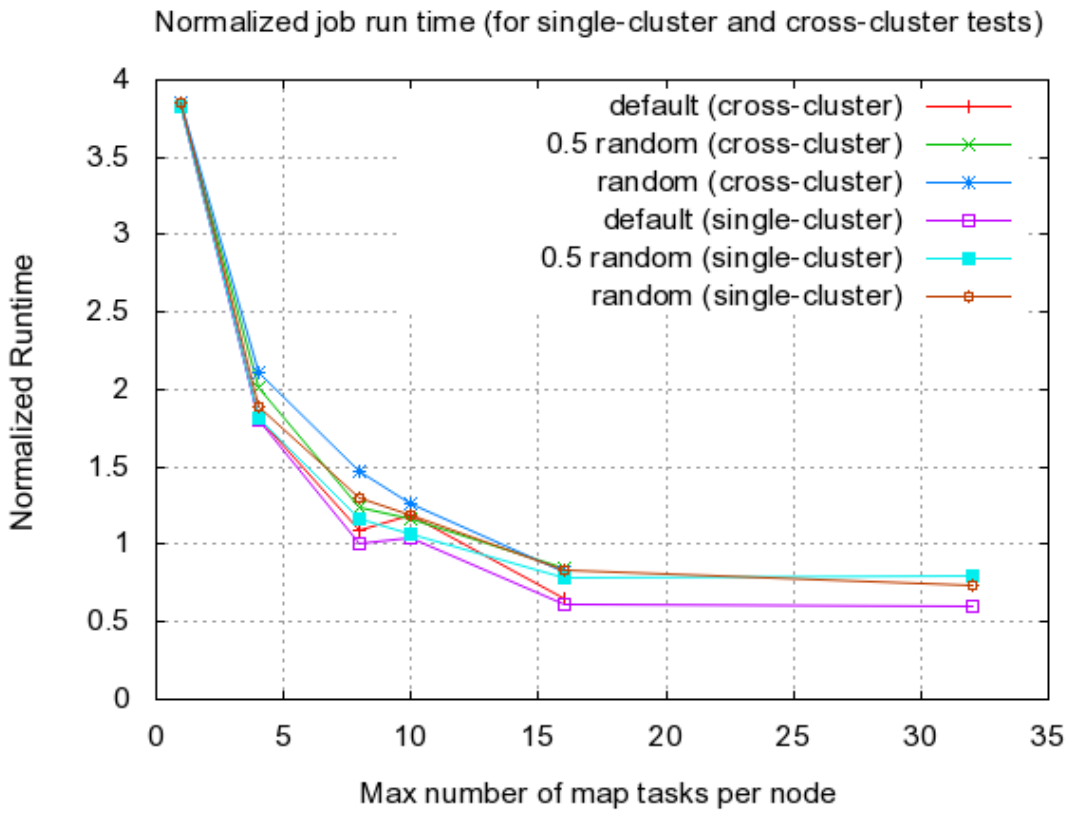


- **Normalized job run time (All results are normalized to the result of the intra-cluster test with 8 mappers):**

Comp
HPC-sty
In my te
comput

!!!Note:
differer

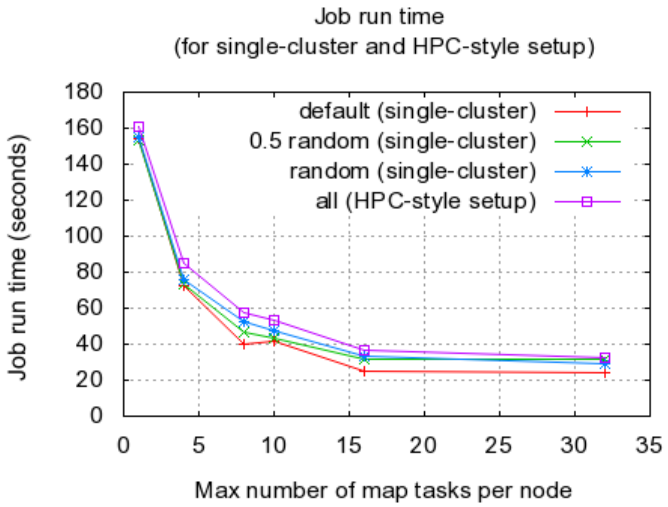
Results



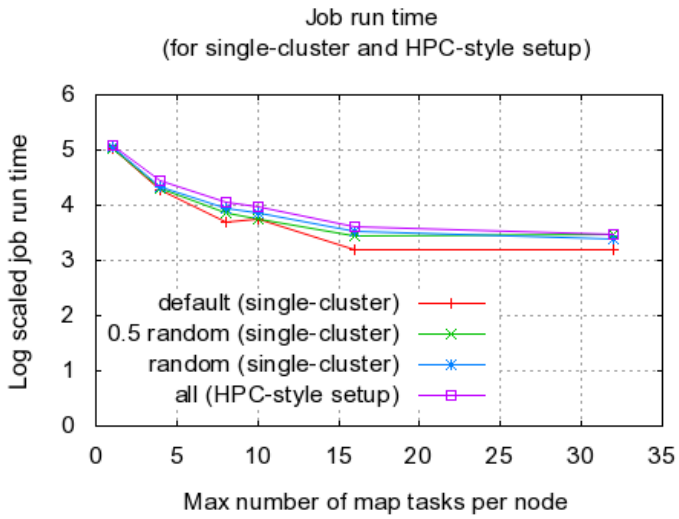
Defaul				
0.5 Rando				
	8	43.1	49	53.2
	10	31.5	30	36.6
	16	32.0		32.2
	32			
Random	1	155	155	160.7
	2	76.1	82	84.9
	4	52.3	53	57.2
	8	47.9	51	53.2
	10	33.6	32	36.6
	16	29.5		32.2
	32			

Following three graphs show the **same results in different ways**.

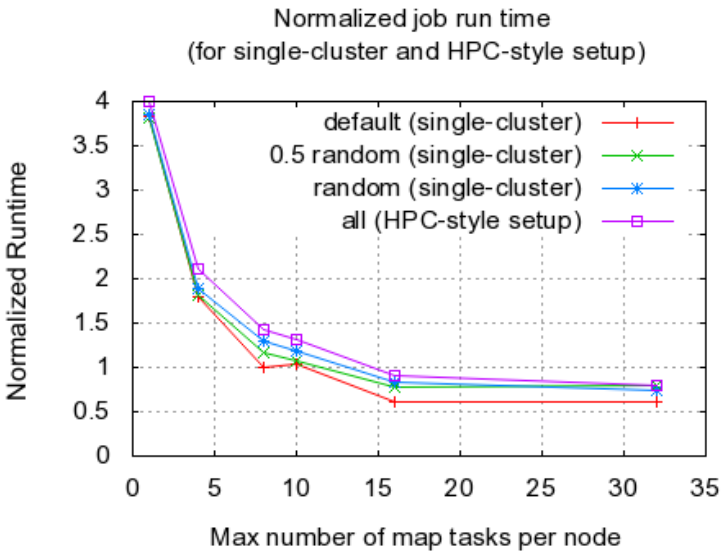
- **Raw job run time**



- **Log-scaled job run time**



- **Normalized job run time**



Summary for cross-cluster tests

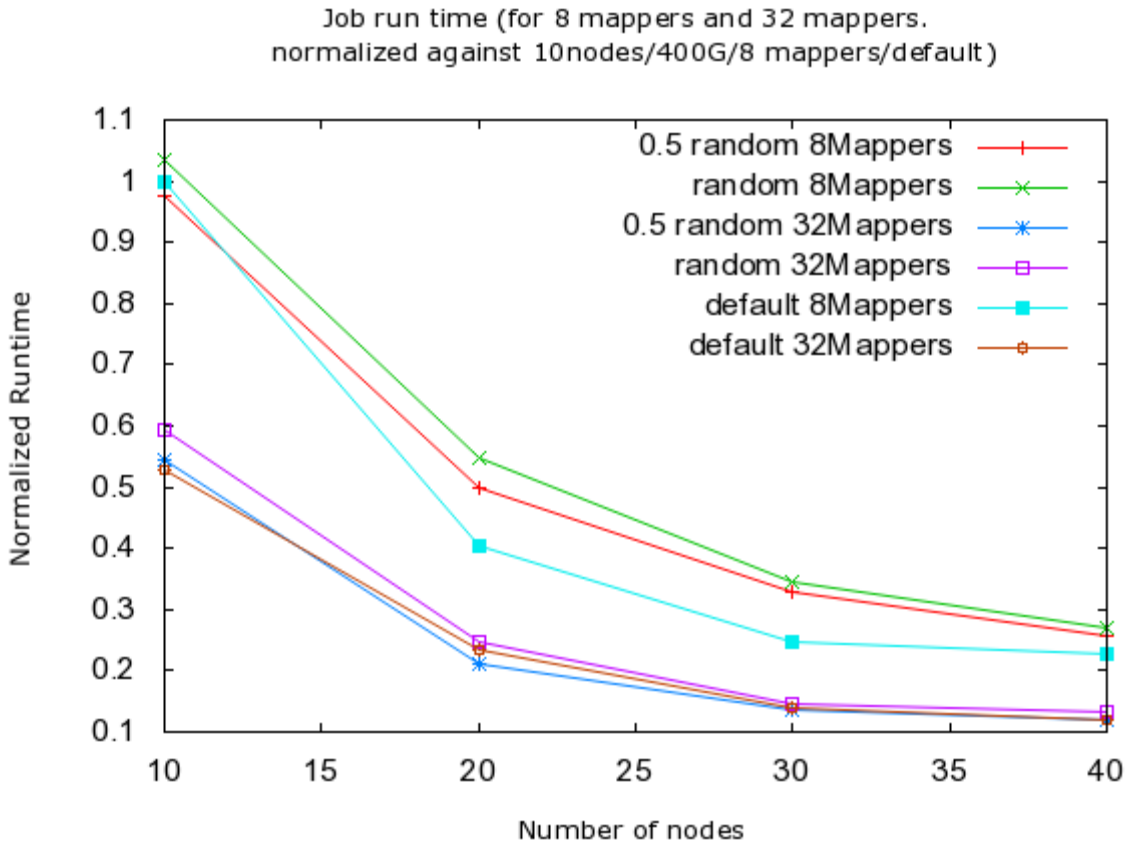
Environment	Slowdown of random scheduling	
VM	20% - 90%	
VM + ViNe	10% - 3000%	
Physical nodes	2% - 20%	

Slowdown of job execution is positively correlated to randomness of scheduling. The more random the scheduling strategy is, the severer performance degradation becomes. Using VM adds additional overhead (25% - 250%). Usually, as max number of map tasks per node is increased, job run time is decreased.

More investigation

- Following graph shows the run time for 8 mappers per node and 32 mappers per node. Data size is fixed to 400GB. Number of nodes varies in the plot. All data is normalized to the result of the test with 10nodes, 400GB input, 8 mappers per node and default scheduling.

  In following graph, the speed up is calculated using: job_runtime/ baseline_runtime


Job run time (for 8 mappers and 32 mappers.
normalized against 10nodes/400G/8 mappers/default)
Normalized Runtime
Number of nodes
0.5 random 8Mappers
random 8Mappers
0.5 random 32Mappers
random 32Mappers
default 8Mappers
default 32Mappers
1.1
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
10
15
20
25
30
35
40