Performance testing

- Required Metrics
    - Client Side Measurements
    - Common Stats
    - GC Log Analysis
    - Server side metrics
    - Log analysis
    - Miscellaneous
- Phase I: Perf Tools
- Phase II: Automation
- Phase III: Correctness
- 0.8 Performance testing
    - Producer throughput

It would be worthwhile to automate our performance testing to act as a generic integration test suite.

The goal of this would be to do basic performance analysis and correctness testing in a distributed environment.

Required Metrics

Client Side Measurements

1. Throughput
2. Response Time/Latency
3. Timeouts
4. Consumer lag

Common Stats

1. Vmstat - Context Switch, User CPU utilization %, System CPU utilization %, Total CPU utilization %
2. Iostat - Reads/sec, Writes/sec, KiloBytes read/sec, KiloBytes write/sec, Average number of transactions waiting, Average number of active transactions, Average response time of transactions, Percent of time waiting for service, Percent of time disk is busy
3. Prstat - Virtual memory size of each java process, RSS size of each process, Total CPU utilization of each process

GC Log Analysis

1. Footprint (Maximal amount of memory allocated)
2. Freed Memory (Total amount of memory that has been freed)
3. Freed Memory/min (Amount of memory that has been freed per minute)
4. Total Time (Time data was collected for)
5. Acc Pauses (Sum of all pauses due to GC)
6. Throughput (Time percentage the application was NOT busy with GC)
7. Full GC Performance (Performance of full GC collections. Full GC collections are marked so in the gc logs.)
8. GC Performance (Performance of minor collections. These are collections that are not full according to the definition above.)
9. CMS counts and frequency (Number of CMS collections and their frequency)
10. CMS failure count and frequency (CMS failure metrics)

Server side metrics

1. Throughput and response time breakdown for each request at the LogManager, RequestPurgatory level
2. ISR membership churn aggregate and per partition
3. Number of expirations in the request purgatory
4. Leader election rate aggregate and per partition
5. Leader election latency aggregate and per partition
6. High watermark change aggregate and per partition
7. Replica lag time and bytes aggregate and per partition
8. Replica fetch throughput and response time aggregate and breakdown at the LogManager, RequestPurgatory level

Log analysis

1. Exceptions in logs, their frequency and types of exception
2. Warnings in logs, their frequency and types of warnings

Miscellaneous

1. Capture all the server machine profiles before tests are being executed (Such as disk space, number of CPUS etc)
2. Capture all configurations for each run

Phase I: Perf Tools

The goal of this phase is just to create tools to help run perf tests. We already have some of these so this will primarily just be about expanding and augmenting these.

- kafka-producer-perf-test.sh - We will add a csv option to this to dump incremental statistics in csv format for consumption by automated tools.
- kafka-consumer-perf-test.sh - Likewise we will add a csv option here.
- jmx-dump.sh - This will just poll the kafka and zookeeper jmx stats every 30 seconds or so and output them as csv.
- dstat - This is the existing perf tool to get read/write/fs stats
- draw-performance-graphs.r - This will be an R script to draw relevant graphs given a bunch of csv files from the above tools. The output of this script will be a bunch of png files that can be combined with some html to act as a perf report.

Here are the graphs that would be good to see:

- Latency histogram for producer
- MB/sec and messages/sec produced
- MB/sec and messages/sec consumed
- Flush time
- Errors (should not be any)
- Consumer cache hit ratio (both the bytes and count, specifically 1 - #physical_reads / #requests and 1 - physical_bytes_read / bytes_read)
- Write merge ratio (num_physical_writes/num_produce_requests and avg_request_size/avg_physical_write_size)
- CPU, network, io, etc

Phase II: Automation

This phase is about automating the deployment and running of the performance tests. At the end of this phase we want to have a script that pulls from svn every night, runs a set of performance scenarios, and produces reporting on these.

We need the following helper scripts for this:

- kafka-deploy-kafka.sh - This script will take a set of hosts and deploy kafka to each of them.
- kafka-start-cluster.sh - This will start the kafka broker on the list of hosts
- kafka-stop-cluster - Stops cluster

The tests will be divided up into scenarios. Each scenario is a directory which contains the following:

- broker config
- producer config
- consumer config
- producer perf test command
- consumer perf test command
- env file that contain # brokers, # producers, and # consumers

The output of the scenario will be a directory which contains the following:

- Producer perf csvs
- Consumer perf csvs
- dstat csvs
- jmx csvs
- env file

Scenarios to test:

- Producer throughput with no consumers. We should cover the following cases:
    - Vary the number of topics
    - Vary the async batch size
    - Vary the flush size
    - Vary the message size
- Consumer throughput with no producer
    - Vary the message size
    - Vary the number of topics
- Single producer/consumer pair
    - Cold consumption (i.e. not in cache)
    - Active consumption (i.e. consumer caught up to producer)
    - Vary the number of topics
- Multiple consumers for one topic

We should add a script to take two scenarios and produce a summary/diff of them, i.e. what go worse and what got better. We can use this to track things over time. We can also rsync these up to a public location as a service to open source developers.

Phase III: Correctness

The correctness testing can be very straight-forward, all we want to validate is that every message produced gets consumed. This could be as simple as logging out a simple message id in the consumer and comparing it to the produced value.

Simplest idea is just to have each producer produce a set of known messages (say sequential integers in some unique range). Then have consumers validate that all integers were consumed (no gaps) and record the number of duplicates (if any).

Ideally we would repeat this scenario and script in broker failures (kills), server pauses (simulated), etc.

0.8 Performance testing

Producer throughput

Message size : ~1K Production Data

Throughput in MB/s

		Kafka Version	0.7		0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8		0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8
		Replication Factor	n/a		1	1	1	1	1	1	1	1	1		2	2	2	2	2	2	2	2	2
		Acks	n/a		-1	1	0	-1	1	0	-1	1	0		-1	1	0	-1	1	0	-1	1	0
		Compression	Uncomp		Uncomp	Uncomp	Uncomp	Gzip	Gzip	Gzip	Snappy	Snappy	Snappy		Uncomp	Uncomp	Uncomp	Gzip	Gzip	Gzip	Snappy	Snappy	Snappy
Producer threads	**Batch size**																						
1	1		29.57		1.66	1.69	19.48	0.94	0.97	3.51	1.53	1.46	11.40		0.56	1.61	23.52	0.35	0.91	3.70	0.56	1.45	10.83
2	1		45.35		3.31	3.11	23.82	1.80	1.72	3.87	2.30	2.96	11.65		1.27	2.79	20.59	0.66	1.39	3.86	1.07	2.19	11.39
5	1		58.53		5.24	5.49	20.26	2.59	2.86	3.35	4.02	4.48	13.36		1.79	4.76	21.29	1.16	2.49	3.06	1.93	3.98	10.49
10	1		50.23		8.20	8.59	19.65	3.17	2.77	2.87	7.35	7.39	10.51		2.90	7.97	19.44	1.85	2.68	3.05	3.27	7.24	12.01
1	50		49.15		17.95	18.88	77.43	7.93	7.66	26.88	15.42	15.44	62.33		10.57	15.76	57.85	4.79	7.50	28.61	10.31	13.98	66.15
2	50		84.24		34.94	33.41	82.67	14.88	15.25	30.13	29.77	26.47	71.18		17.26	27.26	78.50	9.53	14.29	30.11	16.82	26.10	90.98
5	50		102.44		64.38	62.82	89.66	17.56	18.23	19.47	57.54	58.58	71.71		26.02	47.69	86.62	12.48	17.39	31.54	33.13	54.18	74.96
10	50		103.02		61.06	64.57	86.48	18.45	17.24	20.38	59.59	58.44	69.34		28.08	59.80	91.62	18.68	17.90	20.02	35.62	62.03	72.02
1	100		49.76		23.69	23.56	75.71	8.93	9.04	28.69	21.73	21.79	70.55		12.69	21.61	52.12	5.86	8.85	27.49	13.48	21.04	66.70
2	100		84.75		42.85	40.46	84.72	18.48	16.90	31.69	42.61	38.07	85.78		22.54	35.81	74.53	11.16	17.18	35.86	25.63	37.95	106.99
5	100		92.78		67.65	67.89	87.24	21.93	19.78	21.40	67.22	65.94	87.98		26.93	57.77	86.46	15.20	18.92	25.35	37.40	75.18	83.91
10	100		100.23		67.53	68.40	87.99	19.82	20.80	21.38	69.64	68.49	81.88		31.36	63.19	88.68	17.39	18.96	21.56	43.26	72.14	84.89