Test setup: the tests were run locally in the server to minimize connection delays (these were at 0-1ms throughout the tests). Standalone mode has the usual standalone Zope/Plone instance through buildout while Zeo has 4 clients to one Zeo server. The server processor Is 2.0GHz Intel Xeon with 4 cores.

	requests	concurrency	time taken	failed reqs	write errors	reqs/second	time/request (ms)	Transfer rate (kB/s)	connection times: total min-mean-max(**median**)
Zeo	1000	1	136.9	0	0	7.3	136.8	94.5	123-136-738(126)
Standalone			128.5	0	0	7.8	128.6	99.5	122-128-395(124)

Starting values. There's no concurrency so the tests take quite a long. The waiting times are quite low (for Plone!). The standalone instance performs a bit faster propably due to fact that there's no performance taking middleware with it's apache configuration (mod_proxy_balancer)

Zeo	1000	10	36	0	0	27.5	36.3	358	125-360-3646(305)
Standalone			134.5	0	0	7.43	134.5	95	257-1342-57000(1215)

Now we are seeing some difference! The Zeo setup rules standalone 100-0. The standalone performance is similar to first case when there was no concurrency at all. It didn't drop any requests though. But look at the Zeo values: 4 times the requests per second even though the waiting times are 4 times the standalone

Zeo	9000	10	332	0	0	27	36.9	352	126-368-1512(287)
Standalone			this would have taken about 20 minutes so I didn't go for this. I'd trust that 1000 requests is enough to review continual performance						

For Zeo: there were 9x requests compared to earlier and the values are same (even lower) -> I think we can assume that this continual performance.
Let's drop requests count back to 1000

Zeo	1000	100	37.6	0	0	26.6	37.5	346	410-3564-4718(3612)
Standalone			150.4	0	0	6.7	150.4	85	327-14534-141420(6063)

This was expected: Standalone has quite similar performance to earlier but waiting times are like eternity :) The Zeo server can do as many requests/second as earlier but due to 100 concurrent requests the waiting times are a lot longer. Seems like Zeo server has reached it's performance limit via concurrency 10 - theoretically it should be 4 so let's test that next:

Zeo	1000	4	44.6	0	0	22.4	44.6	292	125-177-1202(132)

Hmm, compared to test with 10 concurrency the values are little worse. This indicates that the server retains some preformance reserve which would kick in with heavier pages. I think that practise and theory meet well.

Zeo	1000	1000	apr_socket_recv: Connection reset by peer (104). Total of 687 requests completed
Standalone			apr_socket_recv: Connection reset by peer (104). Total of 300 requests completed

Hehe, I was expecting this. 1000 simultaneous tasks is something you couldn't handle yourself :) Still Zeo performs better in this

For curiosity, let's increase the Zeo clients to 6 and execute one average concurrency test. Theoretically, we shouldn't get any performance boost from this:

Zeo (6 clients)	1000	6	34.8	0	0	28.8	34.8	373.9	126-207-1367(169)
	1000	10	34.7	0	0	28.9	34.7	375	127-345-2670(316)

No differences between these and no difference to Zeo setup with 4 clients either. One can see a very slight raise in request times so 6 client setup performs worse to that of 4 clients. This could be some fluctuating, though, but more propably it's the reason of extra overhead from 6 clients or something like that.

The conclusion: have as many Zeo clients as you have processors / cores in your processor. This was expected but now we have evidence of it.
With very low traffic sites which rarely get more than 1 request per second this doesn't matter but with high-traffic sites which get multiple concurrent requests the Zeo server is clearly more powerful

The performance increase: extra processors/cores x standalone

But what about resource requirements? The used RAM increase for 6 client Zeo setup was whopping 621 MB (1132 MB -> 1753 MB). That means about 100 MB per Zeo client as the Zeo server memory requirement was only about 12-15 MB. Thus, only use as many Zeo clients as absolutely necessary!