Lab Four

1. Document API for library

1.1 Server: server.js

Attribute:

- const WebSocket
- const wss
- var employeeNum: client number
- var startTime = []: record start time of each client
- var endTime = []: record end time of each client
- var executeTime = []: calculate execute time of each client

Method:

- wss.on('connection', (ws, req) => {}): client connect to server event listener
- ws.on('message', message => {}): client send message to server event listener

1.2 Test application: main.cpp

Attribute:

- int workload: workload of employee
- int employeeID: employee's unique id

Method:

- void work(): Add up to workload of employee, after each adding, client will send the updated workload of the employee to server, and wait for server's instruction for continuing next adding.
- void fire(): After reaching target workload, client will be terminated, which means employee is fired.
- EM_BOOL WebSocketOpen(int eventType, const EmscriptenWebSocketOpenEvent *e, void *userData){}: Listen for websocket open event.
- EM_BOOL WebSocketClose(int eventType, const EmscriptenWebSocketOpenEvent *e, void *userData){}: Listen for websocket close event.
- EM_BOOL WebSocketError(int eventType, const EmscriptenWebSocketOpenEvent *e, void *userData){}: Listen for errors in websocket.
- EM_BOOL WebSocketMessage(int eventType, const EmscriptenWebSocketOpenEvent *e, void *userData){}: Listen for message sending event in socket sending.

1.3 Native JS test application: client.js

Attribute:

- var workload: workload of employee
- var employeeID: employee's unique id

Method:

- function work(): workload of employee
- function fire(): employee's unique id
- socket.addEventListener('open', function(msg){}): websocket open event listener
- socket.addEventListener('message', function(e){}): server send message to client event listener

1.4 Shell script to open 10 clients(native js) at a time, running 50 times: test.sh

```
for i in {1..50}
do
    for j in {1..10}
    do
        open test.html
    done
    sleep 40
done
```

1.5 Shell script to open 10 clients(wasm) at a time, running 50 times: wasm.sh

```
for i in {1..50}
do
    for j in {1..10}
    do
        open wasm.html
    done
    sleep 40
done
```

2. Design Concept

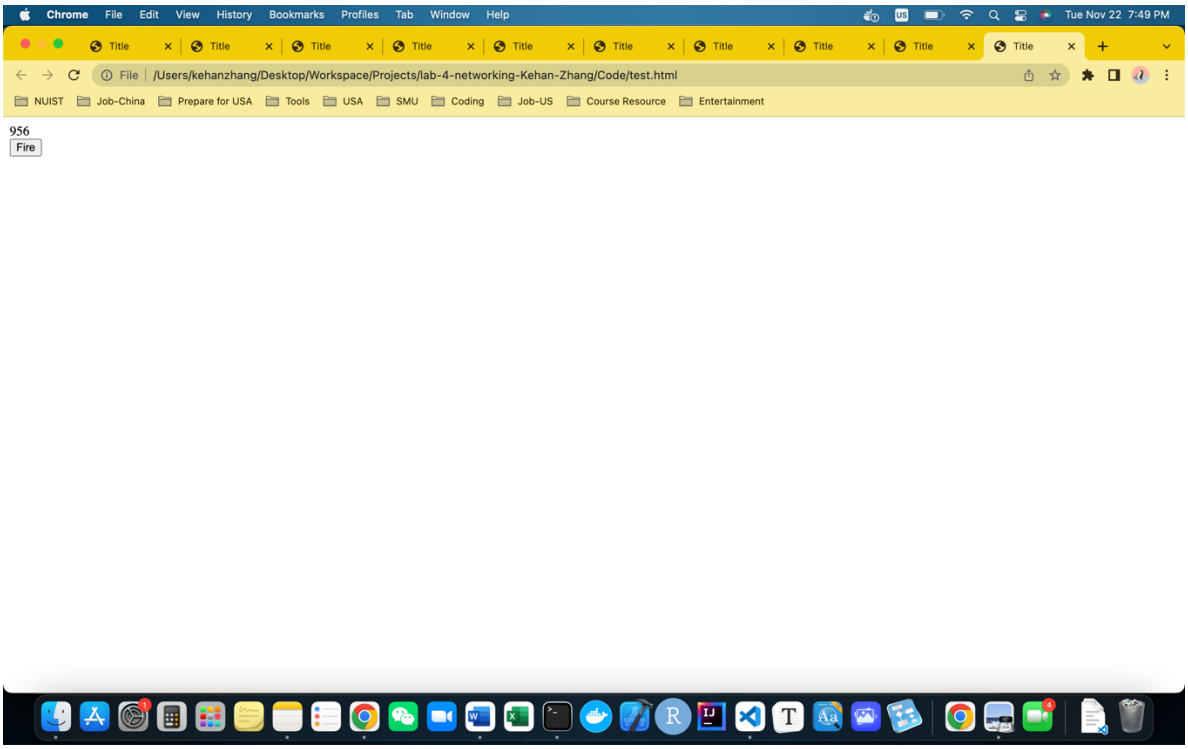
2.1 Workflow

First use 'node server.js' to start the server

```
kehanzhang@KEHANs-MBP Code % node server.js
Server Online
```

Then, run the shell script to open 10 clients at the same time

```
kehanzhang@KEHANs-MacBook-Pro Code % time bash test.sh
bash test.sh  0.24s user 0.19s system 1% cpu 42.580 total
```



If click the button fire on the web page, this employee will be fired. Otherwise, it will continues to work until its workload reach 5000 then be fired. After being fired, client will send the work report to server, including employee id and total workload.

```
Server Message Recived: {"id":10,"type":"stat","data":"5000"}
```

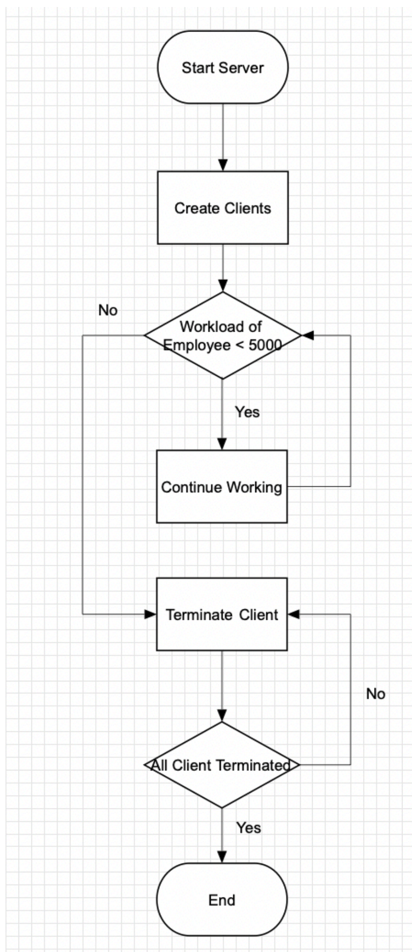
Each time employee complete one task(workload + 1), it will send a message to the server, including its unique employee id and its current workload.

```
Server Message Recived: {"id":8,"type":"work","data":4863}
Server Message Recived: {"id":9,"type":"work","data":4827}
Server Message Recived: {"id":7,"type":"work","data":4954}
Server Message Recived: {"id":10,"type":"work","data":4359}
Server Message Recived: {"id":8,"type":"work","data":4864}
Server Message Recived: {"id":7,"type":"work","data":4955}
Server Message Recived: {"id":10,"type":"work","data":4360}
Server Message Recived: {"id":9,"type":"work","data":4828}
Server Message Recived: {"id":8,"type":"work","data":4865}
Server Message Recived: {"id":10,"type":"work","data":4361}
Server Message Recived: {"id":9,"type":"work","data":4829}
Server Message Recived: {"id":7,"type":"work","data":4956}
Server Message Recived: {"id":8,"type":"work","data":4866}
Server Message Recived: {"id":10,"type":"work","data":4362}
Server Message Recived: {"id":9,"type":"work","data":4830}
Server Message Recived: {"id":7,"type":"work","data":4957}
Server Message Recived: {"id":8,"type":"work","data":4867}
Server Message Recived: {"id":9,"type":"work","data":4831}
Server Message Recived: {"id":10,"type":"work","data":4363}
Server Message Recived: {"id":7,"type":"work","data":4958}
Server Message Recived: {"id":8,"type":"work","data":4868}
Server Message Recived: {"id":9,"type":"work","data":4832}
Server Message Recived: {"id":10,"type":"work","data":4364}
Server Message Recived: {"id":7,"type":"work","data":4959}
Server Message Recived: {"id":8,"type":"work","data":4869}
```

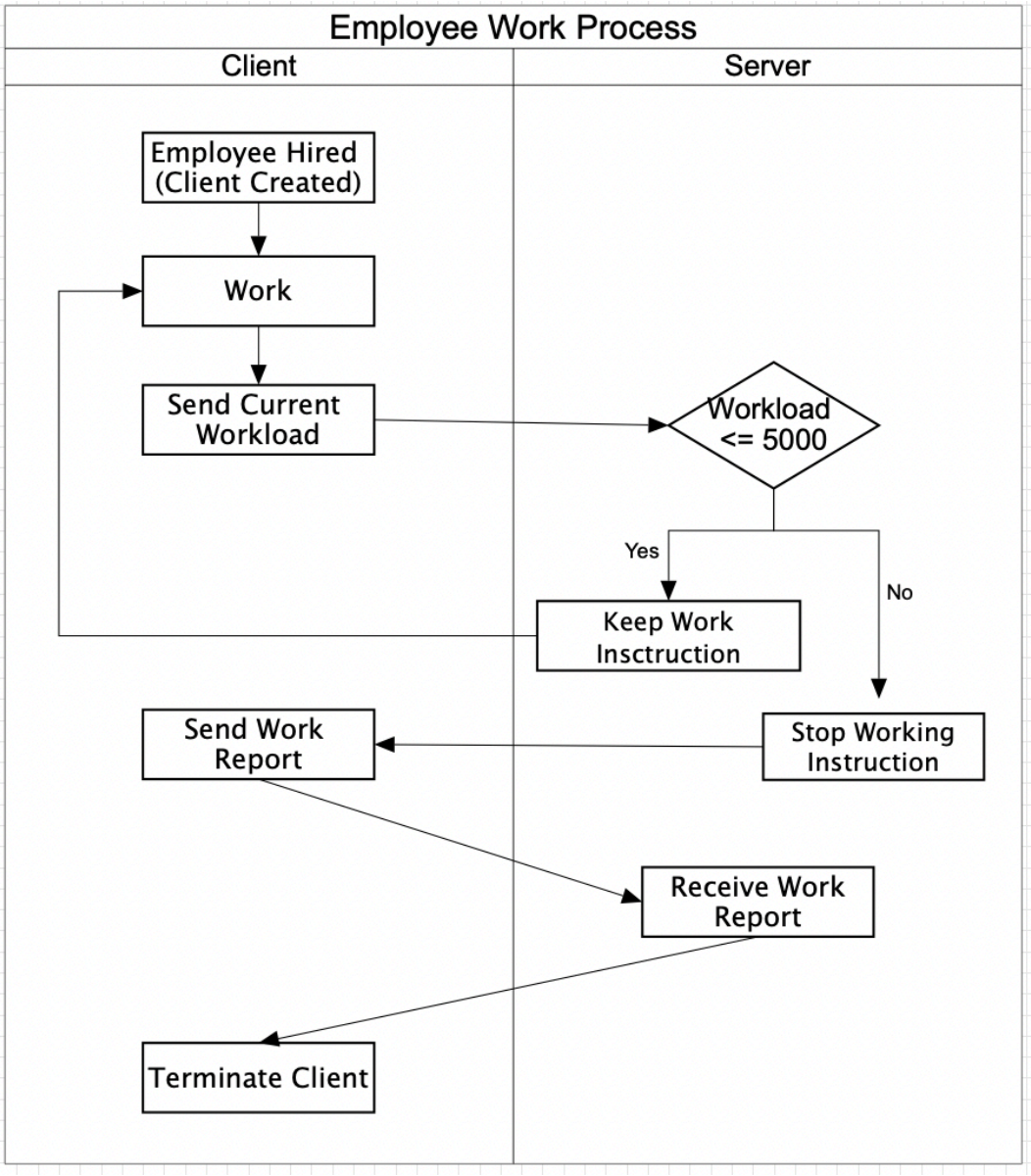
Server will then check if the workload of employee has reached 5000. If not, server will send "work" instruction to client, which indicates that employee has to keep working. Otherwise, employee will be fired and client will be terminated.

```
Client Message Recived: {"type":"work"}                    client.js:12
Client Message Recived: {"type":"work"}                    client.js:12
Client Message Recived: {"type":"work"}                    client.js:12
Client Message Recived: {"type":"work"}                    client.js:12
Client Message Recived: {"type":"work"}                    client.js:12
Client Message Recived: {"type":"work"}                    client.js:12
Client Message Recived: {"type":"work"}                    client.js:12
Client Message Recived: {"type":"work"}                    client.js:12
Client Message Recived: {"type":"work"}                    client.js:12
Client Message Recived: {"type":"work"}                    client.js:12
Client Message Recived: {"type":"work"}                    client.js:12
Client Message Recived: {"type":"work"}                    client.js:12
Client Message Recived: {"type":"work"}                    client.js:12
Client Message Recived: {"type":"work"}                    client.js:12
Client Message Recived: {"type":"work"}                    client.js:12
Client Message Recived: {"type":"work"}                    client.js:12
Client Message Recived: {"type":"work"}                    client.js:12
Client Message Recived: {"type":"work"}                    client.js:12
Client Message Recived: {"type":"work"}                    client.js:12
Client Message Recived: {"type":"work"}                    client.js:12
Client Message Recived: {"type":"work"}                    client.js:12
Client Message Recived: {"type":"work"}                    client.js:12
Client Message Recived: {"type":"work"}                    client.js:12
Client Message Recived: {"type":"work"}                    client.js:12
Client Message Recived: {"type":"work"}                    client.js:12
```

Flowchart of this library



2.2 Statechart Diagram



3. Compare Execution Time between Native JS and WASM

In this part, I will create 2 clients, 10 clients and 20 clients in both Native JS and WASM for 50 time. Compare the execution runtime of Native JS (2 clients) and WASM (2 clients), Native JS (10 clients) and WASM (10 clients), Native JS (20 clients) and WASM (20 clients).

1. Show 50 execution runtime of Native JS and WASM to one excel file, and perform basic statistic measure on them.

WASM (10 Clients)	Native JS (10 Clients)	WASM (2 Clients)	Native JS (2 Clients)	WASM (20 Clients)	Native JS (20 Clients)
23.18830069	20.68120447	5.270801044	4.707620768	38.0005168	33.704689
22.45771525	20.37521437	4.834178683	4.27239975	36.61293692	36.48895963
22.2112805	18.8560061	4.660158538	4.171066118	34.08226186	35.21838647
22.30948627	18.7387091	5.107907913	3.9587154	34.54114323	37.98532354
23.39125722	18.26634389	4.693861558	4.125017382	34.96024759	33.35369789
23.20361016	22.0969946	4.991511797	5.286210018	37.49528818	31.89021804
24.60118402	20.18116378	4.899300996	4.471200905	34.33980402	32.86300019
22.3608311	20.48881115	5.32350756	4.214979858	34.57991853	34.557088
23.42578992	18.57519499	4.692483186	4.084300476	35.14791462	36.28513537
22.02282863	18.69289511	4.832396506	4.34913229	34.79356666	34.55018733
22.96078885	18.75198675	5.585785055	4.295797578	36.42289392	37.30225866
22.56113938	21.38841833	4.864049881	4.305260958	33.95031803	32.95243447
22.71437474	18.89471508	4.893415613	4.297684744	33.8687426	33.29674703
23.71514356	20.33315838	4.838239796	4.262697725	34.29508519	33.75203126
23.0988727	19.02300445	4.718360541	4.325089381	34.87571039	35.60665702
22.28068777	21.09604339	5.007759876	4.216470005	35.52020463	35.90325077
22.32589266	19.25453757	5.139536325	4.2228281	34.15944445	32.41716787
22.92691556	19.70248327	5.099376662	4.454217494	36.34831417	37.02425812
22.29709498	19.06985798	4.788669458	4.107092795	35.25101943	36.70692867
23.01139139	18.70377303	4.7793384	4.184665758	34.95466469	36.30886797
22.40149336	18.09763345	4.723779487	4.088092662	37.27753793	34.23667627
22.88105133	19.70335769	4.739047383	4.100658966	36.66038583	34.4489227
23.11290986	18.45571396	4.685854843	4.638340637	36.16781628	35.52385468
23.3682562	18.58342545	4.938465081	4.356680134	35.57750798	35.13808226
22.60907981	18.78442611	4.8674857	4.164428656	35.01971758	34.49846198
22.58118038	19.95154296	4.795313394	4.196051342	35.35125952	33.75261285
22.26635998	18.76918534	4.926813613	4.048706813	34.31802101	33.18166605
24.19900022	22.23667704	4.972896144	4.294204483	36.35796838	33.31166443
23.62694735	18.15652725	5.099526523	4.436466669	36.35271004	38.2802455

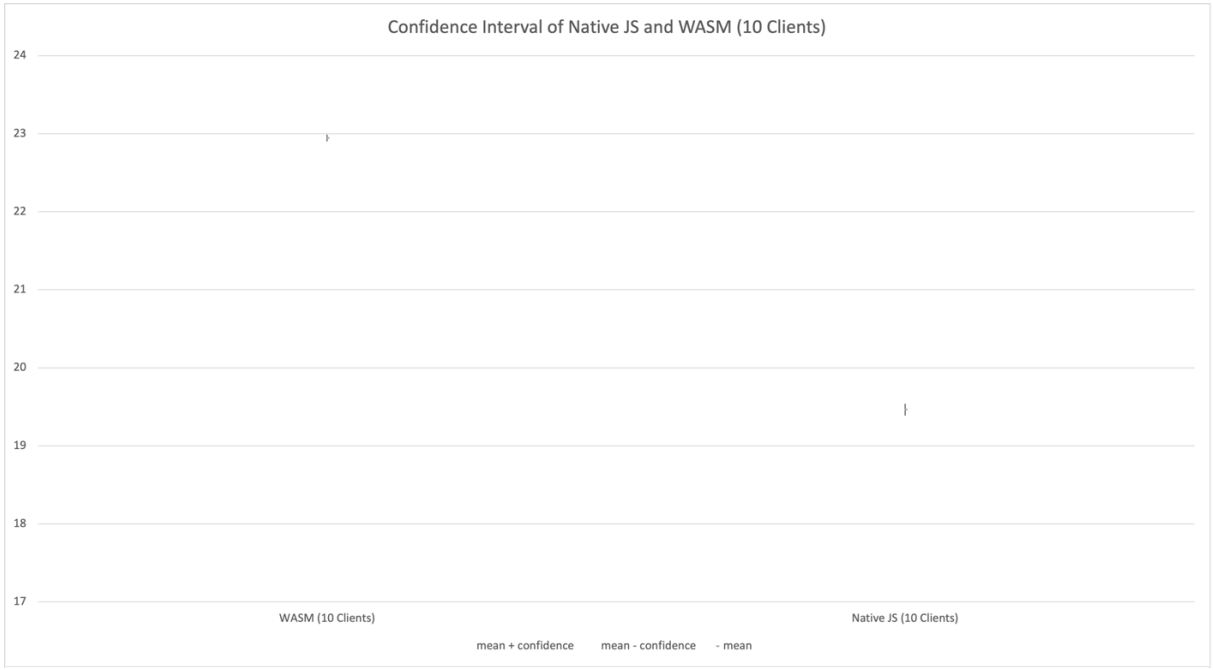
We can tell from the statistic results in the form below that average execution time of WASM is always higher than Native JS in 2 clients, 10 clients and 20 clients. What is more, execution time of 20 clients is higher than 10 clients, 10 clients is higher than 2 clients running either in Native JS and WASM.

	WASM (10 Clients)	Native JS (10 Clients)	WASM (2 Clients)	Native JS (2 Clients)	WASM (20 Clients)	Native JS (20 Clients)
mean	22.94532358	19.46377091	4.957382951	4.274103956	35.48443852	34.96345529
max	24.69084865	22.23667704	5.72921751	5.286210018	38.0005168	38.29567653
min	21.48394143	16.86402673	4.652916641	3.9587154	33.8687426	31.89021804

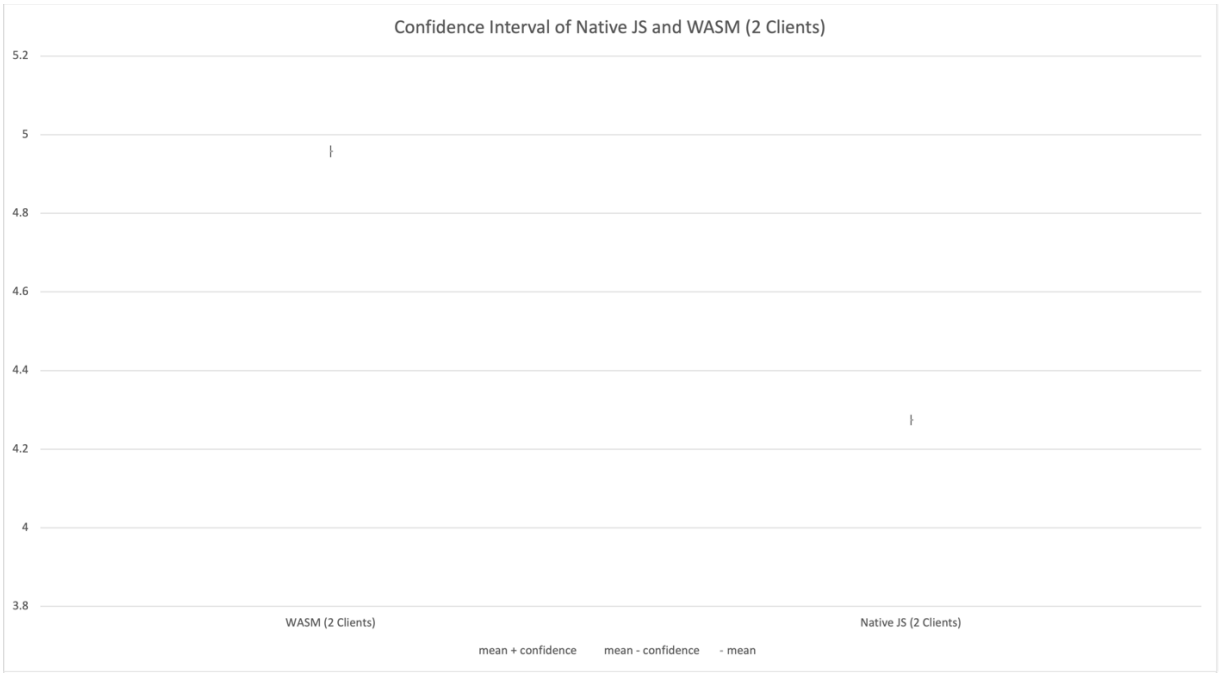
2. Use Excel to calculate 95% confidence intervals.

	WASM (10 Clients)	Native JS (10 Clients)	WASM (2 Clients)	Native JS (2 Clients)	WASM (20 Clients)	Native JS (20 Clients)
mean	22.94532358	19.46377091	4.957382951	4.274103956	35.48443852	34.96345529
max	24.69084865	22.23667704	5.72921751	5.286210018	38.0005168	38.29567653
min	21.48394143	16.86402673	4.652916641	3.9587154	33.8687426	31.89021804
standard_dev	0.694502419	1.221827066	0.240249491	0.216268522	1.055922733	1.597761382
95% confidence	0.043044915	0.075728235	0.014890544	0.013404216	0.065445566	0.099028457
mean - confidence	22.90227867	19.38804268	4.942492407	4.260699741	35.41899295	34.86442684
mean + confidence	22.98836849	19.53949915	4.972273495	4.287508172	35.54988409	35.06248375

For 10 clients, from the following graph we can tell that lower bound of confidence interval of WASM is higher than the upper bound of confidence interval of Native JS, which means there is no overlap between confidence interval of WASM and Native JS. Therefore, we can conclude that these results are of statistically significant, performance of Native JS is better than WASM when we run 10 clients.



For 2 clients, from the following graph we can tell that lower bound of confidence interval of WASM is higher than the upper bound of confidence interval of Native JS, which means there is no overlap between confidence interval of WASM and Native JS. Therefore, we can conclude that these results are of statistically significant, performance of Native JS is better than WASM when we run 2 clients.



For 20 clients, from the following graph we can tell that lower bound of confidence interval of WASM is higher than the upper bound of confidence interval of Native JS, which means there is no overlap between confidence interval of WASM and Native JS. Therefore, we can conclude that these results are of statistically significant, performance of Native JS is better than WASM when we run 20 clients.