# Joint Speed Scaling and Sleep Management for Power Efficient Computing

**Yanpei Liu** [1] Aman Chadha [1] Stark C. Draper [2] Nam Sung Kim [1]

[1]Electrical and Computer Engineering
University of Wisconsin Madison

[2]Electrical and Computer Engineering
University of Toronto

DIMACS Workshop on Algorithms for Green Data Storage
Dec 18, 2013

# Acknowledgment


- Murali Annavaram (University of Southern California)
- Ken Vu (IBM)
- Srinivasan Ramani (IBM)
- Thomas Wenisch (University of Michigan Ann Arbor)

# Motivating facts

DIMACS Workshop on Algorithms for Green Data Storage

# Backgrounds

- Two important power control methods.
  - Speed scaling and low-power states.
  - Are often exploited in separation.
    - ⋆ Speed scaling: *[GH01][ALW10][DMR11][BMB12]*.
    - ⋆ ON/OFF: *[MGW09][GHA10][N11]*.
  - Should be jointly optimized, managed and operated.

| | |
|---|---|
| $C0_{(i)}$ | Operating idle state: there is no work to do, voltage & frequency held constant at last DVFS setting |
| $C1$ | Halt state: clock stops |
| $C3$ | Sleep state: cache flushed, architectural state maintained, clock stopped |
| $C6$ | Deep sleep state: architectural state saved to RAM, voltage set to zero |

# Challenges

Challenge 1:

- Suppose we have a low utilization server.
- Given two low-power states in idle:
  - **Shallow sleep**: quick wake up and power hungry.
  - **Deep sleep**: slow wake up and power efficient.

- *If the response time must be kept low,* **shallow sleep** *or* **deep sleep***?*
- *If the response time is okay to be high,* **shallow sleep** *or* **deep sleep***?*

Challenge 2:

- Suppose a CPU has many low-power states.

- *Should we concatenate then all?*

# Queuing-theoretic analysis

- Model a single server as $M/G/1$ queue. Arrival rate $\lambda$, operating frequency $f \in [0,1]$ (DVFS), service rate $\mu f$ and utilization $\rho = \lambda/\mu$.
- When busy, run at frequency $f$, incurring power $P_0 f^3 + C$.
    - Example: $P_0 = 130$ *Watts* and $C = 112$ *Watts*.
- When idle: enter $n$ low-power states.
    - The system enters *ith* low-power state $\tau_i$ seconds after its queue empties, $\tau_1 \leq \tau_2 \leq \tau_3 \ldots \leq \tau_n$.
    - Power at *ith* low power state is $P_i$, $P_1 > P_2 > \ldots > P_n$.
    - Wake-up latency is $w_i$ (with power), $w_1 < w_2 < \ldots < w_n$.

| $C0_{(i)}$ | $C1$ | $C3$ | $C6$ |
|---|---|---|---|
| – | – | – | – |
| $0\ s$ | $1-10\ \mu s$ | $10-100\ \mu s$ | $0.1-1\ ms$ |
| – | – | – | $1-10\ s$ |

- With $n = 1$, $f = 1$, $\tau_1 = 0$, it reduces to the well-known "race-to-halt" mechanism.

## Theoretical results – power

- $P_i$: power at state $i$. $\tau_i$: entrance delay for state $i$. $w_i$: wakeup latency for state $i$, $f$: frequency, $\mu$: service rate and $\lambda$: arrival rate.

### Theorem

*The average power consumption for an $M/M/1$ single-server system with $n$ low-power states is*

$$\mathbb{E}[P] = \frac{1}{\lambda L}\left[\sum_{i=1}^{n-1} P_i\left(e^{-\lambda\tau_i} - e^{-\lambda\tau_{i+1}}\right) + P_n e^{-\lambda\tau_n}\right] + P_0\left(1 - \frac{e^{-\lambda\tau_1}}{\lambda L}\right) \tag{1}$$

*where $L$ is defined as*

$$L = \frac{\mu f + \mu f \lambda\left[\sum_{i=1}^{n-1} w_i\left(e^{-\lambda\tau_i} - e^{-\lambda\tau_{i+1}}\right) + w_n e^{-\lambda\tau_n}\right]}{\lambda(\mu f - \lambda)}. \tag{2}$$

# Theoretical results – mean response time

## Theorem

*The mean response time for an $M/M/1$ server system with $n$ low power states is*

$$\mathbb{E}[R] = \frac{1}{\mu f - \lambda} + \frac{2\mathbb{E}[D] + \lambda \mathbb{E}[D^2]}{2(1 + \lambda \mathbb{E}[D])}, \tag{3}$$

*where*

$$\mathbb{E}[D] = \sum_{i=1}^{n-1} w_i (e^{-\lambda \tau_i} - e^{-\lambda \tau_{i+1}}) + w_n e^{-\lambda \tau_n}, \tag{4}$$

$$\mathbb{E}[D^2] = \sum_{i=1}^{n-1} w_i^2 (e^{-\lambda \tau_i} - e^{-\lambda \tau_{i+1}}) + w_n^2 e^{-\lambda \tau_n}. \tag{5}$$

# Theoretical results – deadline

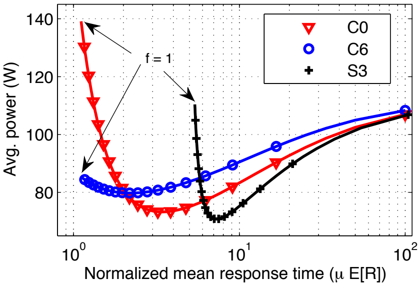
- Special case when $n = 1, \tau_1 = 0$.

**Theorem**

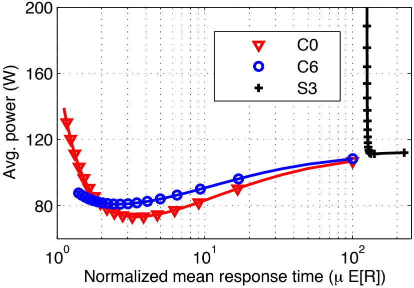
*The probability for the response time to exceed a deadline $Pr(R \geq d)$ for an $M/M/1$ single-server is*

$$Pr(R \geq d) = \frac{e^{-(\mu f - \lambda)d} - w_1(\mu f - \lambda)e^{-d/w_1}}{1 - w_1(\mu f - \lambda)}. \quad (6)$$

# Engineering lesson I – low utilization



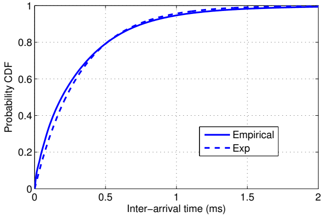

(a) DNS (194 *ms*): $\rho = \lambda/\mu = 0.1$.



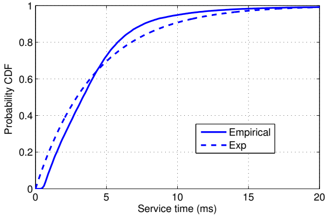

(b) Google (4.2 *ms*): $\rho = \lambda/\mu = 0.1$.

- There exists optimal frequency $f$.
  - Too fast causes power to increase. Too slow takes longer to finish.
- The best power state depends on the response time constraint.
  - Tight: deep sleep (blue). Loose: shallow sleep (red).

# Engineering lesson I – low utilization



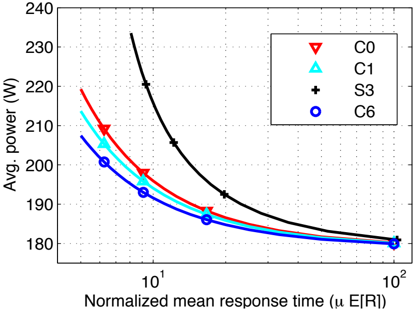

(c) Google inter-arrival time.



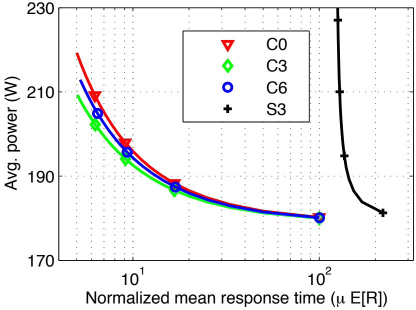

(d) Google service time.

Figure 1: Statistics of Google workload [MWW 12].

# Engineering lesson II – high utilization



(a) DNS (194 *ms*): $\rho = 0.8$.

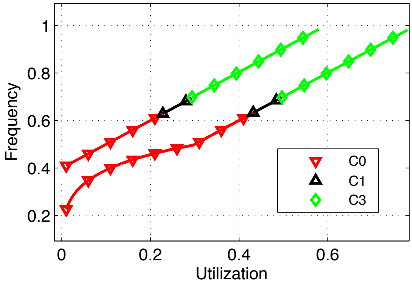


(b) Google (4.2 *ms*): $\rho = 0.8$.

- Power saving comes mostly from performance scaling.
  - Rarely enter low-power states.
- Optimal policy is job size dependent.
  - Large jobs can tolerate more wake up latency.

# Engineering lesson III – best policies

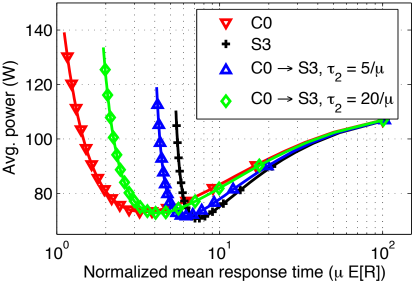
- What do best polices look like at different utilization?




(c) Google $\mathbb{E}[R]$ constraint.

- No "one-size-fits-all" policy.
  - Different policies should be used under different utilization.
- "Bump" at low utilization
  - Caused by the slack in the quality-of-service.

# Engineering lesson IV – delayed entrance




(d) DNS (194 *ms*): delayed $S3$ at $\rho = 0.1$.

- Optimal performance scaling and entrance delay combination.
- Sequential power throttle-back may be conservative.
    - High utilization: rarely enters the last state. Low utilization, waste to not enter the optimal state.

# Conclusion

Thank you