Proposing a Replacement for FreeBSD's powerd (Preview) Or, how I tamed the fan of my notebook

Dominic Fandrey

Von Leitner-Institut für verteiltes Echtzeit-Java

GPN16, May 2016

		Challenges!	
Contents			

Who?



	Challenges!	

Contents

Who?

What?



		Challenges!	
Contents	5		

Who?

What?

Why?



Who?	What?	Why?	Challenges!	How?	Conclusions!
Contents	5				
Who?					
What?					
Why?					
Challer	ıges!				



Who?	١	What?	Why?	Challenges!	How?	Conclusions!
Cor	ntents					
	1001100					
	Who?					
	What?					
	Why?					
	Challenges!					
	enditen.geo.					
	How?					

Who	?	What?	Why?	Challenges!	How?	Conclusions!
Сс	ontents					
	Who?					
	What?					
	Why?					
	Challenges!					
	How?					
	Conclusions	1				
	Conclusions)! 				
						Access

Who?		Challenges!	

- Dominic Fandrey <kami@freebsd.org>
- M.Sc. (Computer Science)



Who?		Challenges!	

- Dominic Fandrey <kami@freebsd.org>
- M.Sc. (Computer Science)
- Located in Europe/Karlsruhe



Who?		Challenges!	

- Dominic Fandrey <kami@freebsd.org>
- M.Sc. (Computer Science)
- Located in Europe/Karlsruhe
- Working as a researcher at an undisclosed polytechnic university





Who?		Challenges!	

- Dominic Fandrey <kami@freebsd.org>
- M.Sc. (Computer Science)
- Located in Europe/Karlsruhe
- Working as a researcher at an undisclosed polytechnic university







Who?		Challenges!	

- Dominic Fandrey <kami@freebsd.org>
- M.Sc. (Computer Science)
- Located in Europe/Karlsruhe
- Working as a researcher at an undisclosed polytechnic university









Who?		Challenges!	

- Dominic Fandrey <kami@freebsd.org>
- M.Sc. (Computer Science)
- Located in Europe/Karlsruhe
- Working as a researcher at an undisclosed polytechnic university











What?	Challenges!	

► Load:



	What?	Challenges!	
Definiti	ons		

► Load:

▶ The fraction of CPU cycles not spent idle



What?	Challenges!	

Load:

- ► The fraction of CPU cycles not spent idle
- P-State:



What?	Challenges!	

- ► Load:
 - The fraction of CPU cycles not spent idle
- P-State:
 - Performance State, also frequently called stepping



What?	Challenges!	

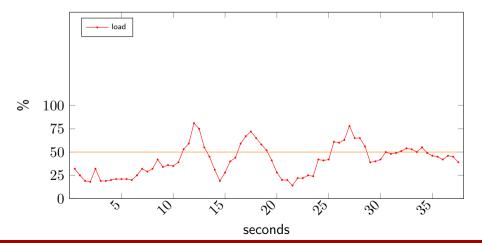
- Load:
 - The fraction of CPU cycles not spent idle
- P-State:
 - Performance State, also frequently called stepping
 - ► A CPU mode of operation with a specific clock frequency and core voltage



What?	Challenges!	

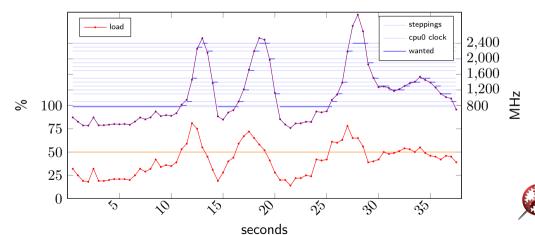
CPU p-state control

powerd++ adaptive



What?	Challenges!	

CPU p-state control

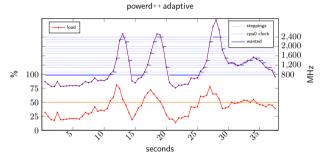


 $powerd {\scriptstyle ++} \, adaptive$

	Why?	Challenges!	

Why control p-state?

► Fan noise





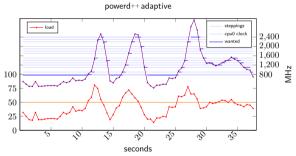
	Why?	Challenges!	

%

Why control p-state?

► Fan noise

Battery/Energy conservation

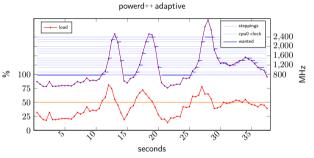




	Why?	Challenges!	

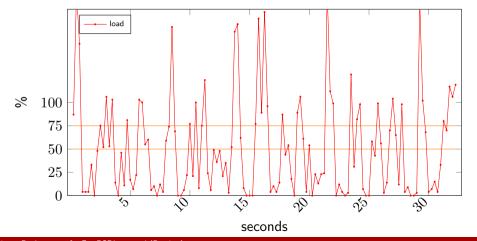
Why control p-state?

- ► Fan noise
- Battery/Energy conservation
- Hardware lifetime

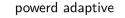


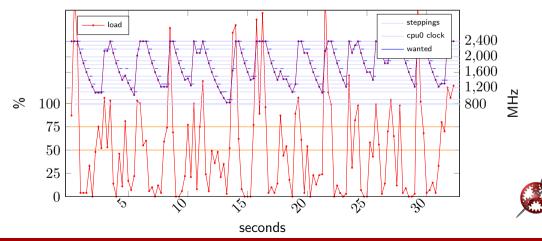
	Why?	Challenges!	

powerd adaptive

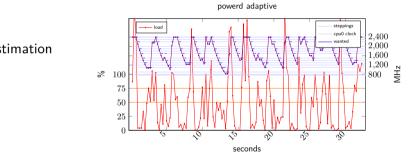


	Why?	Challenges!	





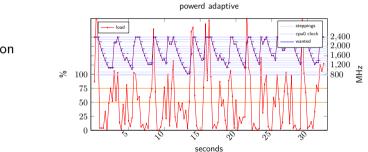
	Why?	Challenges!	



Broken load estimation



	Why?	Challenges!	



Broken load estimation

Aggressive speeding



	Why?	Challenges!	

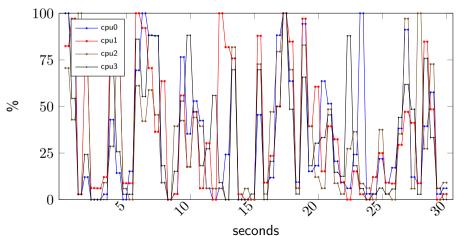
- powerd adaptive steppings load cpu0 clock 2,400wanted 2,000 1,600 1,200 MHz % 100800 7550250 হ 20 3 seconds
- Broken load estimation
- Aggressive speeding
- Reluctant braking

	Why?	Challenges!	

- powerd adaptive steppings load cpu0 clock 2,400wanted 2,0001,600 1,200 MHz % 100800 755025হ 20 3 seconds
- Broken load estimation
- Aggressive speeding
- Reluctant braking
- Excessive fan noise



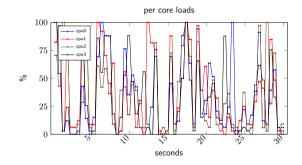
	Challenges!	



per core loads

	Challenges!	

Load is noisy

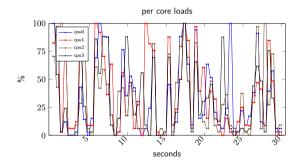




	Challenges!	

Load is noisy

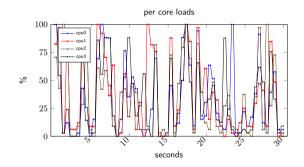
Load shifts





	Challenges!	

- Load is noisy
- Load shifts
- Load saturates





	Challenges!	

Control algorithm

Contradicting goals:



	Challenges!	

Control algorithm

- Contradicting goals:
 - System should be responsive



	Challenges!	

Control algorithm

- Contradicting goals:
 - System should be responsive
 - System should be energy efficient



	Challenges!	

- Contradicting goals:
 - System should be responsive
 - System should be energy efficient
- P-States:



	Challenges!	

- Contradicting goals:
 - System should be responsive
 - System should be energy efficient
- P-States:
 - P-States lie



	Challenges!	

- Contradicting goals:
 - System should be responsive
 - System should be energy efficient
- P-States:
 - P-States lie
 - Extremely low nonsense p-states



	Challenges!	

- Contradicting goals:
 - System should be responsive
 - System should be energy efficient
- P-States:
 - P-States lie
 - Extremely low nonsense p-states
 - P-State switching has a cost



	Challenges!	

- Contradicting goals:
 - System should be responsive
 - System should be energy efficient
- P-States:
 - P-States lie
 - Extremely low nonsense p-states
 - P-State switching has a cost

Conclusions:



	Challenges!	

- Contradicting goals:
 - System should be responsive
 - System should be energy efficient
- P-States:
 - P-States lie
 - Extremely low nonsense p-states
 - P-State switching has a cost

Conclusions:

 Make the right compromises



	Challenges!	

- Contradicting goals:
 - System should be responsive
 - System should be energy efficient
- P-States:
 - P-States lie
 - Extremely low nonsense p-states
 - P-State switching has a cost

- Conclusions:
 - Make the right compromises
 - Make it tunable



	Challenges!	

- Contradicting goals:
 - System should be responsive
 - System should be energy efficient
- P-States:
 - P-States lie
 - Extremely low nonsense p-states
 - P-State switching has a cost

- Conclusions:
 - Make the right compromises
 - Make it tunable
 - Set sane defaults



		Challenges!	How?	
Summary				

powerd



		Challenges!	How?	
Summary				

powerd

Any granularity p-state changes



		Challenges!	How?	
Summary				

Any granularity p-state changes

powerd

 Global p-state changes only



		Challenges!	How?	Conclusions!
Summary				

- Any granularity p-state changes
- Maximum load

powerd

 Global p-state changes only



		Challenges!	How?	
<u> </u>				
Summar	у			

- Any granularity p-state changes
- Maximum load

powerd

- Global p-state changes only
- Sum of loads



		Challenges!	How?	
Summar	ry -			

- Any granularity p-state changes
- Maximum load
- ► Aim for a load target

versus

- Global p-state changes only
- Sum of loads



		Challenges!	How?	
Summar				

- Any granularity p-state changes
- Maximum load
- Aim for a load target

versus

- Global p-state changes only
- Sum of loads
- Hysteresis



		Challenges!	How?	
Summar				

powerd++

- Any granularity p-state changes
- Maximum load
- Aim for a load target
- Filter load to ignore short spikes

- Global p-state changes only
- Sum of loads
- Hysteresis



		Challenges!	How?	
-				
Summar	Y			

powerd++

- Any granularity p-state changes
- Maximum load
- Aim for a load target
- Filter load to ignore short spikes

- Global p-state changes only
- Sum of loads
- Hysteresis
 - Aggressively tuned for responsiveness



		Challenges!	How?	Conclusions!
Summary				

powerd++

- Any granularity p-state changes
- Maximum load
- Aim for a load target
- Filter load to ignore short spikes
- Explicit CLA syntax like
 --max 1.2ghz

- Global p-state changes only
- Sum of loads
- Hysteresis
 - Aggressively tuned for responsiveness



		Challenges!	How?	Conclusions!
Summary				

powerd++

- Any granularity p-state changes
- Maximum load
- Aim for a load target
- Filter load to ignore short spikes
- Explicit CLA syntax like
 --max 1.2ghz

- Global p-state changes only
- Sum of loads
- Hysteresis
 - Aggressively tuned for responsiveness
 - ▶ Hard coded units -M 1200



	Challenges!	How?	

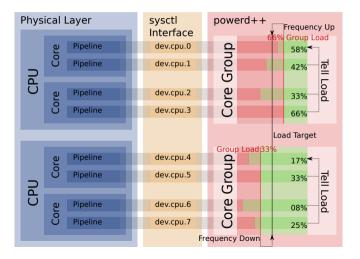
Control algorithm (powerd)

Phys	ical I	_ayer	sysctl		power			
			Interfa	ice		Overall Loa	d	283%
	Core	Pipeline	dev.cp	u.0		58%		
	ŭ	Pipeline	dev.cp	u.1		42%		
CPU								
0	Core	Pipeline	dev.cp	u.2		33%		
	ပိ	Pipeline	dev.cp	u.3		66%		
	Core	Pipeline	dev.cp	u.4		17%		
_	ပိ	Pipeline	dev.cp	u.5		33%	75%	î Step up
CPU							-	steresis
Ū	Core	Pipeline	dev.cp	u.6		08%	50%	↓Step do.
	ů	Pipeline	dev.cp	u.7		25%		



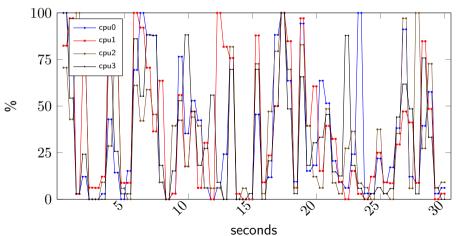
	Challenges!	How?	

Control algorithm (powerd++)





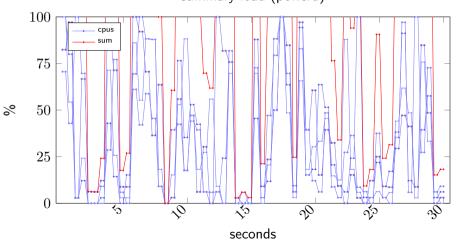
	Challenges!	How?	



per core loads



	Challenges!	How?	

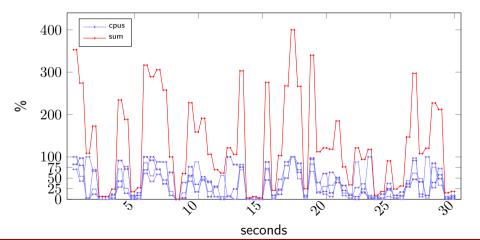


summary load (powerd)



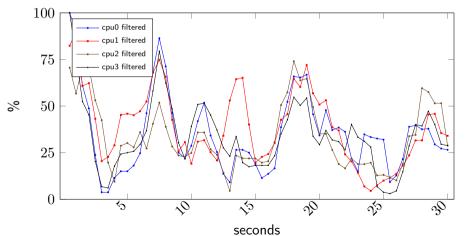
	Challenges!	How?	

summary load (powerd)



	Challenges!	How?	

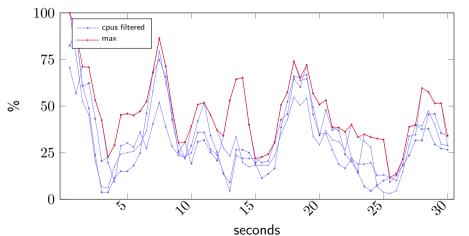
filtered loads (powerd++)





	Challenges!	How?	

max of filtered load (powerd++)



	Challenges!	Conclusions!

Load signal noise



	Challenges!	Conclusions!

Load signal noise

Lower sample rate, gliding average



	Challenges!	Conclusions!

- Load signal noise
- Low multi core loads

Lower sample rate, gliding average



	Challenges!	Conclusions!

- Load signal noise
- Low multi core loads

- Lower sample rate, gliding average
- Without breaking single core loads



	Challenges!	Conclusions!

High frequency core hopping



	Challenges!	Conclusions!

► High frequency core hopping

► This is rare



	Challenges!	Conclusions!

- High frequency core hopping
- P-States that lie reduce accuracy

This is rare



	Challenges!	Conclusions!

- High frequency core hopping
- P-States that lie reduce accuracy

- This is rare
- Ignoring this works well enough



	Challenges!	Conclusions!

- High frequency core hopping
- P-States that lie reduce accuracy
- kern_clock.c only supports global frequency changes

- This is rare
- Ignoring this works well enough



	Challenges!	Conclusions!

- High frequency core hopping
- P-States that lie reduce accuracy
- kern_clock.c only supports global frequency changes

- This is rare
- Ignoring this works well enough
- This is fixable, but may break scheduling



	Challenges!	Conclusions!

$\setminus (-)/$ Praise the sun!

https://github.com/lonkamikaze/powerdxx

