به نام خدا

میانترم سیستم های نهفته

جناب آقای دکتر انصاری

سارا آذرنوش

98170668

2)

Checkpoint :

چک پوینت یک موضوع مهم در محاسبات با تحمل خطا به عنوان مبنایی برای بازیابی است. برای مثال یک کاربر در حال انجام یک محاسبات طولانی مدت است و به دلایلی (سخت افزار یا نرم افزار)، ماشینی که محاسبه میکند شکست می خورد. اگر checkpoint نباشد و برنامه تمام شود، تمام محاسبات قبلی از بین میرود. اگر کاربر برنامه را ذخیره کرده بود می توانست برنامه را از آخرین بازرسی راه اندازی مجدد کند. این بازگشت به یک checkpoint ذخیره شده نامیده می شود.

1) چک پوینت تکنیکی است که تحمل خطا را برای سیستم های محاسباتی فراهم می کند. اساساً شامل ذخیره یک image فوری از وضعیت برنامه است، به طوری که برنامه ها می توانند در صورت خرابی از آن نقطه راه اندازی مجدد شوند. این امر به ویژه برای برنامه های کاربردی طولانی که در سیستم های محاسباتی مستعد شکست اجرا می شوند، مهم است.

2) چک پوینت نوعی تکنیک برای گنجاندن تحمل خطا در یک سیستم است.

3) فرآیند ذخیره وضعیت یک فرآیند یا وظیفه است.

در ادامه قرار دادن checkpoint ها را مطالعه می‌کنیم تا بازیابی از شکست‌ها را بدون از دست دادن ددلاین تضمین کنیم. و دو حالت uniform و nonuniform را که دو مدل از چک پوینتیگ هستند را مقایسه میکنیم.

دو متغیر که از آن ها در ادامه بسیار استفاده میشوندσ و ρ هستند.

σ: The higher the value of σ the less the amount of slack available.

ρ: the overhead of checkpointing relative to the total available time to execute

uniform

به طور شهودی، هرچه نقاط بیشتری checkpoint گرفته شود، کار کمتری در معرض خطر است و بنابراین، بخشی از slack که برای بازیابی برگشتی رزرو شده است، کوچک‌تر است و امکان استفاده از slack باقی‌مانده را فراهم می‌کند. با این حال برای کاهش بیشتر سرعت پردازنده سربار checkpoint بخشی از slack موجود را مصرف می کند که برای کاهش سرعت استفاده می شود. در نتیجه، هرچه checkpoints بیشتر گرفته شود، فرصت کاهش سرعت کمتر می شود.

شکل 2 انرژی مصرف شده را با توجه به تعداد checkpointمتفاوت نشان میدهد. شکل چند نکته جالب را نشان می دهد. اول، حداقل مصرف انرژی معمولاً در مقادیر غیر صحیح n به دست می آید. دوم، تعداد بهینه checkpoint معمولاً کم است و در صورت افزایش تعداد checkpoint ، مصرف انرژی افزایش زیادی پیدا می‌کند.

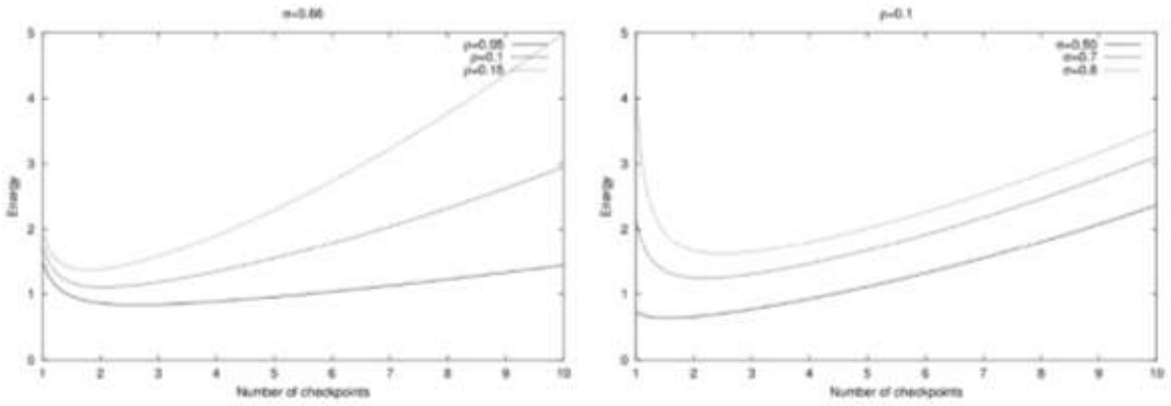

Figure 2: Energy consumption Vs. the number of checkpoints, for fixed $\sigma = 0.66$ (left) and fixed $\rho = 0.1$ (right)

در شکل 3 سطح انرژی را برای تعداد بهینه checkpoint به دست آمده از $n = \frac{\sigma}{4}(3 + \sqrt{9 + \frac{8}{\rho}})$ نشان میدهد. شکل برای مقایسه مقادیر مدیریت انرژی با و بدون تحمل خطا است و نشان می دهد که انرژی مصرف شده با افزایش σ افزایش می یابد، زیرا سستی کمتری در سیستم وجود دارد و بنابراین فرصت های کمتری برای کاهش سرعت پردازنده وجود دارد. علاوه بر این، با افزایش سربار checkpoint ، مصرف انرژی افزایش می یابد. برخی از منحنی ها (با مصرف انرژی بیشتر) "complete" نیستند: زیرا مقادیر بالاتر σ و ρراه حل های غیرقابل اجرا به دست می میآورند. شکل همچنین مصرف انرژی را با فرض اینکه هیچ checkpointی گرفته نشده است نشان می دهد (NoFT).

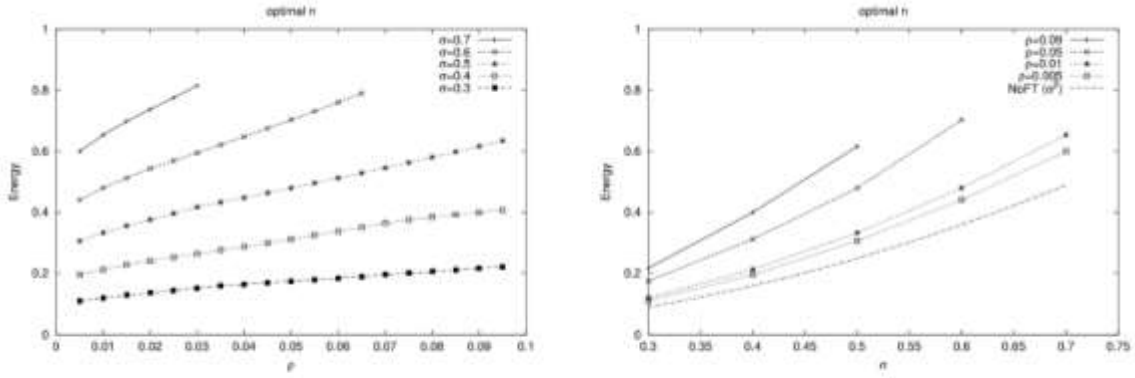

Figure 3: Energy consumption for the optimal number of checkpoints, as a function of ρ (left) and σ (right)

جدول 1 تعداد نقاط بازرسی گرفته شده را در دو سناریو و برای ترکیب های مختلف مقادیر برای σ و و ρنشان می دهد. در سناریوی اول که با ft-only نشان داده شده است هاcheckpoint فقط برای پشتیبانی از بازیابی گرفته می‌شوند، اما هیچ مدیریت انرژی صورت نمی‌گیرد. در دوم با ft +ec نشان داده شده است، checkpoint برای پشتیبانی از قابلیت اطمینان و سرعت کنترل می شود تا انرژی صرفه جویی شود. انتخاب $\rho = .005$ ممکن است خیلی کم به نظر برسد، اما برای مقادیر کوچک σاین با یک سربار checkpoint معقول مطابقت دارد ورودی‌های خالی در جدول، موقعیت‌هایی را نشان می‌دهند که در آن‌ها نمی‌توان از چک‌پوینت استفاده کرد، زیرا ترکیبی از slack موجود و سربار چک‌پوینت، تضمین بازیابی در مهلت مقرر را غیرممکن می‌کند، حتی زمانی که با حداکثر سرعت اجرا می‌شود.

جدول دو روند قابل پیش‌بینی را نشان می‌دهد، همانطور که انتظار می‌رود، زمانی که slack نسبتاً بزرگ باشد (مثلا 0.3)، می‌توان با افزودن checkpoint صرفه‌جویی قابل‌توجهی در انرژی به دست آورد. با این حال، با کاهش slack، کاهش سرعت باعث صرفه جویی کمتر انرژی می شود. تا زمانی که پس انداز عملاً در 0.8 ناپدید شود. این به دلیل ناکارآمدی کاهش سرعت نیست، بلکه به این دلیل است که salck حتی به منظور بازیابی به تنهایی امکان بازرسی را نیز ندارد، یعنی صرفه جویی بسیار کمتری در مصرف انرژی دارد. روند دوم این است که کارایی checkpoint بسیار مهم است. هنگامی که سربار زیاد است(0.1 یا بزرگتر)، صرفه جویی در مصرف انرژی بسیار کم است و با مقادیر بزرگتر σبه سرعت ناپدید می شود.

		$\sigma = 0.3$	$\sigma = 0.4$	$\sigma = 0.5$	$\sigma = 0.6$	$\sigma = 0.7$	$\sigma = 0.8$
$\rho = 0.005$	FT-Only	1	1	2	2	3	5
	FT + EC	3	4	5	6	8	9
	Energy Saving	64%	52%	40%	28%	16%	5%
$\rho = 0.01$	FT-Only	1	1	2	2	3	6
	FT + EC	2	3	4	5	6	6
	Energy Saving	61%	48%	35%	22%	10%	0%
$\rho = 0.03$	FT-Only	1	1	2	2	4	-
	FT + EC	2	2	3	3	4	-
	Energy Saving	57%	38%	25%	10%	0%	-
$\rho = 0.05$	FT-Only	1	1	2	2	-	-
	FT + EC	1	2	2	2	-	-
	Energy Saving	50%	30%	20%	0%	-	-
$\rho = 0.07$	FT-Only	1	1	2	-	-	-
	FT + EC	1	2	2	-	-	-
	Energy Saving	47%	22%	15%	-	-	-
$\rho = 0.10$	FT-Only	1	1	2	-	-	-
	FT + EC	1	1	2	-	-	-
	Energy Saving	42%	17%	7%	-	-	-

Table 1: Number of checkpoints when they are taken only for reliability (FT-only), and when they are also taken for energy management (FT+EC).

Non uniform

در این بخش، Non uniform تحلیل میشود. ابتدا تنظیمات سرعت CPU را به عنوان تابعی از تعداد checkpoint نشان داده میشود (شکل 5). زمانی که slack ارائه شده توسط سیستم کم است (یعنی مقدار بالای σ)، CPU معمولاً با سرعت بالایی تنظیم می شود. زیرا نیاز به جبران checkpoint است که slack انجام می شود. از سوی دیگر، با افزایش slack، سرعت پردازنده به عنوان تابعی از تعداد checkpoint کاهش می‌یابد و سپس به سرعت افزایش می‌یابد. انتظار داریم بیشترین دستاوردها در تنظیمات حداقل سرعت باشد. همچنین می‌توانیم ببینیم که هر چه slack در سیستم بیشتر باشد، checkpoint بیشتری را می‌توان گرفت (الزاماً نیست). زمانی که 0.6 و 0.5 و $0.4 = \sigma$برابر است، حداکثر تعداد checkpoint که امکان سنجی برنامه را حفظ می کند، به ترتیب 5، 8 و 11 است (یعنی منحنی ها از آن نقاط فراتر نمی روند). به همین ترتیب، کمترین تعداد checkpoint نیز محدود شده است.

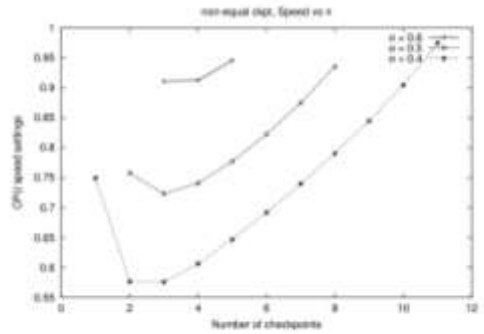

Figure 5: Speed settings as a function of number of checkpoints, for fixed $\rho = 0.05$

در شکل 6 رفتار شهودی بالا مشاهده نمی شود: بیشترین صرفه جویی در انرژی همیشه زمانی نیست که سرعت CPU روی کمترین مقدار تنظیم شود. خط بالایی مربوط به کوچکترین slack در سیستم است و می توان دید که انرژی مصرفی تقریباً به صورت خطی با تعداد checkpoint افزایش می یابد. به این دلیل که اگر checkpoint بیشتری انجام شود، نیاز به افزایش سرعت CPU وجود دارد و زمان کمتری برای محاسبات برای استفاده از CPU، لازم خواهد بود. منحنی $\sigma = 0.5$ که در زمان n=3 سرعت کمتری دارد اما به دلیل checkpoint ، در زمان n=2 انرژی کمتری دارد.

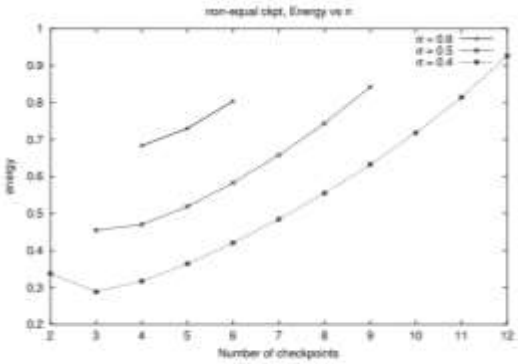

Figure 6: Resulting energy as a function of number of checkpoints, for fixed $\rho = 0.05$

حال uniform را با non uniform مقایسه می کنیم. جدول 2 مقایسه ای بین دو checkpointing placement policies را نشان می دهد. جدول تعداد checkpointرا در non-uniform و ft-only نشان می دهد. جدول همچنین صرفه جویی در انرژی ناشی از non-uniform را در مقایسه با صرفه جویی اضافی که از طرح non-uniform در مقایسه با uniform است را نشان می دهد. جدول نشان می دهد که طرح غیر یکنواخت (non-uniform) در کاهش انرژی بسیار موثر است و تا 68 درصد به کاهش می رسد. مقایسه نشان می دهد که non-uniformنیز در حفظ انرژی موثرتر از uniform است. additional savings در بهترین حالت به حدود 8٪ رسیده است، علاوه بر صرفه جویی که می توان با checkpoint

یکسان به دست آورد. بنابراین، نتیجه می‌گیریم که تکیه بر احتمال کم slack و اتخاذ موضع تهاجمی‌تر، حفاظت بهتری نسبت به رویکرد غیرفعال‌تر دارد.

(Therefore, we conclude that relying on the low probability of faults and taking a more aggressive stance yield better conservation than a more passive approach.)

		$\sigma = 0.3$	$\sigma = 0.4$	$\sigma = 0.5$	$\sigma = 0.6$	$\sigma = 0.7$	$\sigma = 0.8$
$\rho = 0.005$	FT-Only	1	1	2	2	3	5
	Non-Uniform	3	4	5	6	8	9
	Energy Saving	68%	56%	45%	33%	24%	13%
	Diff. w/ Uniform	+5%	+5%	+5%	+4%	+8%	+8%
$\rho = 0.01$	FT-Only	1	1	2	2	3	6
	Non-Uniform	2	3	4	5	6	7
	Energy Saving	65%	53%	42%	29%	16%	3%
	Diff. w/ Uniform	+4%	+5%	+6%	+6%	+6%	+3%
$\rho = 0.03$	FT-Only	1	1	2	2	4	-
	Non-Uniform	2	2	3	3	4	-
	Energy Saving	58%	44%	32%	15%	0%	-
	Diff. w/ Uniform	+4%	+6%	+7%	+5%	+0%	-
$\rho = 0.05$	FT-Only	1	1	2	2	-	-
	Non-Uniform	2	2	2	2	-	-
	Energy Saving	51%	36%	25%	3%	-	-
	Diff. w/ Uniform	+1%	+6%	+5%	+3%	-	-
$\rho = 0.07$	FT-Only	1	1	2	-	-	-
	Non-Uniform	1	2	2	-	-	-
	Energy Saving	47%	28%	19%	-	-	-
	Diff. w/ Uniform	+0%	+6%	+4%	-	-	-
$\rho = 0.10$	FT-Only	1	1	2	-	-	-
	Non-Uniform	1	1	2	-	-	-
	Energy Saving	43%	18%	10%	-	-	-
	Diff. w/ Uniform	+0%	+2%	+3%	-	-	-

Table 2: Number of checkpoints under uniform vs. non-uniform checkpointing, and percentage improvement in energy saving when using non-uniform instead of uniform checkpointing.

در واقع، بین توان مصرفی و قابلیت بازیابی خطا یک معاوضه وجود دارد و تحلیل آماری این مبادله برای سیستم‌های وظیفه عمومی به دلیل تأثیر متقابل بین پارامترهای زیادی که ممکن است بر نتیجه تأثیر بگذارد مفید نیست. با این حال، برای یک سیستم وظیفه معین(given task system) (به عنوان مثال یکی از سیستم های نهفته)، می توان این دو طرح را با هم مقایسه کرد و با توجه به قابلیت اطمینان و مصرف توان مورد نیاز، مناسب ترین را انتخاب کرد. (مانند شکل 7)

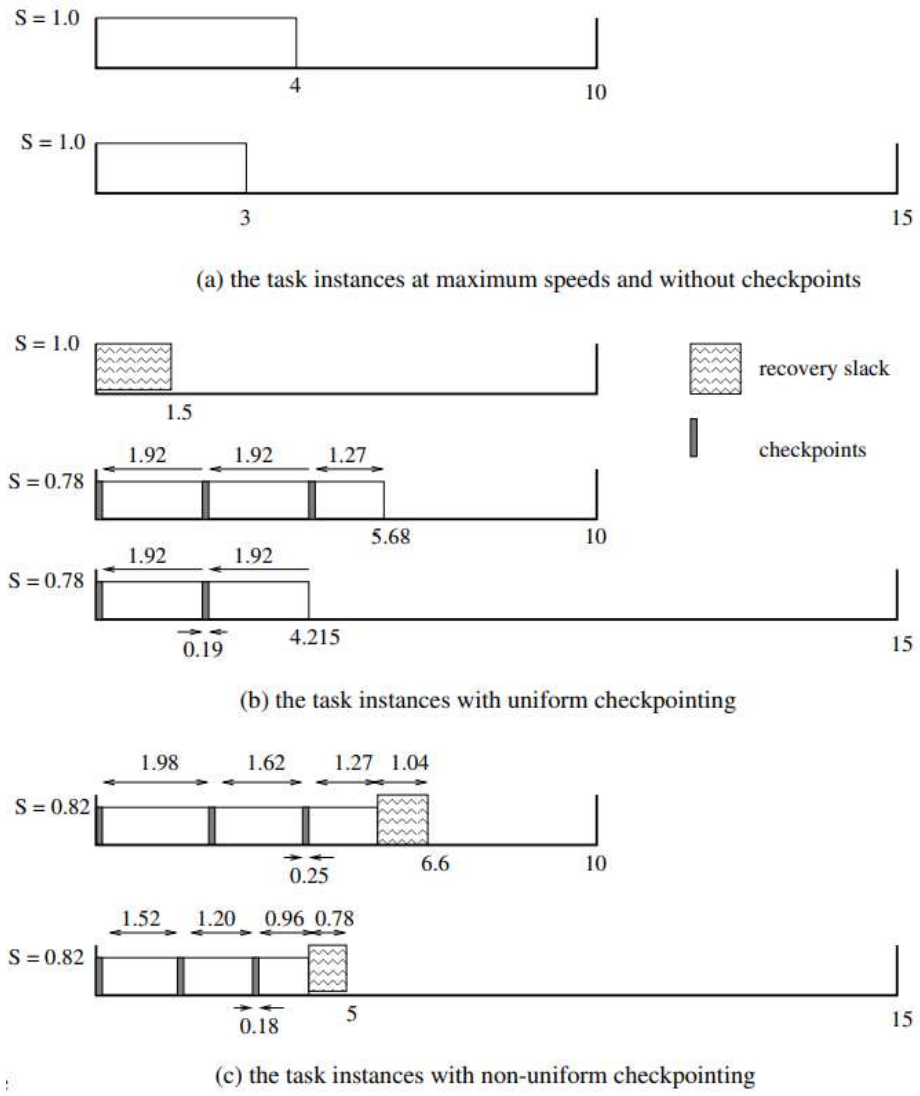

Figure 7: An example of a two-task system

در نهایت، برای مقایسه، بدون مدیریت توان و بدون چک پوینت، مصرف انرژی در طول یک دوره LCM برابر با 0.6است. اگر دو چک پوینت به هر کار اضافه شود، که امکان بازگشت پس از خطا در هر نمونه کار را فراهم کند و سیستم با حداکثر سرعت اجرا شود (بدون مدیریت توان)، سپس مصرف انرژی در طول یک دوره LCM به 65. افزایش می یابد. اگر هیچ چک پوینتی گرفته نشود و کل slack برای کاهش سرعت CPU استفاده شود، مصرف انرژی در طول یک دوره LCM به 36. کاهش می یابد.

	Energy	Recovery capability
No checkpoints and no speed management	0.6 LCM	none
No checkpoints and speed management	0.36 LCM	none
Checkpointing and no speed management	0.65 LCM	one fault per task instance
Uniform checkpoints and speed management	0.52 LCM	one fault every 15 time units
Non-uniform checkpoints and speed management	0.56 LCM	one fault per task instance

Table 3: Energy consumptions (per LCM) and recovery capabilities for the two-task example.

این دو checkpoint به یک سیستم بلادرنگ اجازه می‌دهد تا از خرابی بهبود یابد و مصرف برق را کاهش دهد. هر دو آن ها کاهش سرعت پردازنده را به سطحی که حداقل مصرف انرژی را در حین کار بدون خرابی به همراه دارد، کاهش می‌دهند. اگر خرابی رخ دهد، پردازنده کار از دست رفته را با حداکثر سرعت مجدداً اجرا می کند تا بازیابی را تضمین کند و ددلاین های کار را رعایت کند.

اولین خط policy ، checkpoint را به طور یکنواخت در داخل یک وظیفه قرار می دهد. خط policy دوم این موضع را اتخاذ می کند که از آنجایی که خرابی ها نادر هستند، قابل قبول است که پردازنده را به محض وقوع خرابی و تا پایان مهلت در حداکثر سرعت قرار دهیم و نه فقط در هنگام بازیابی خرابی. این باعث صرفه جویی در انرژی برای برخی از بارهای کاری می شود، اما مستلزم قرار دادن non-uniform است.

منابع

- https://en.wikipedia.org/wiki/Application_checkpointing
- https://www.igi-global.com/dictionary/checkpointing/3741
- Diskless Checkpointing by James S. Plank y, Kai Li x, Michael Puening z, y Department of Computer Science University of Tennessee, x Department of Computer Science Princeton University
- The Interplay of Power Management and Fault Recovery in Real-Time Systems