Aplicaciones MPI resilientes

María J. Martín

Grupo de Arquitectura de Computadores
Universidade da Coruña
mariam@udc.es

COMPUTER ARCHITECTURE GROUP
UNIVERSITY OF A CORUÑA
GAC.UDC.ES

UNIVERSIDADE DA CORUÑA

Contexto

Supercomputadores actuales

- Clusters de procesadores multinúcleo/GPUs
- Alto poder de cómputo $\sim 10^{15}$ FLOPS
- Miles de nodos y miles/millones de núcleos

Summit: Nº 1 lista top 500

- 36.000 procesadores
- Más de 2 millones de núcleos
- 200 PFLOPS

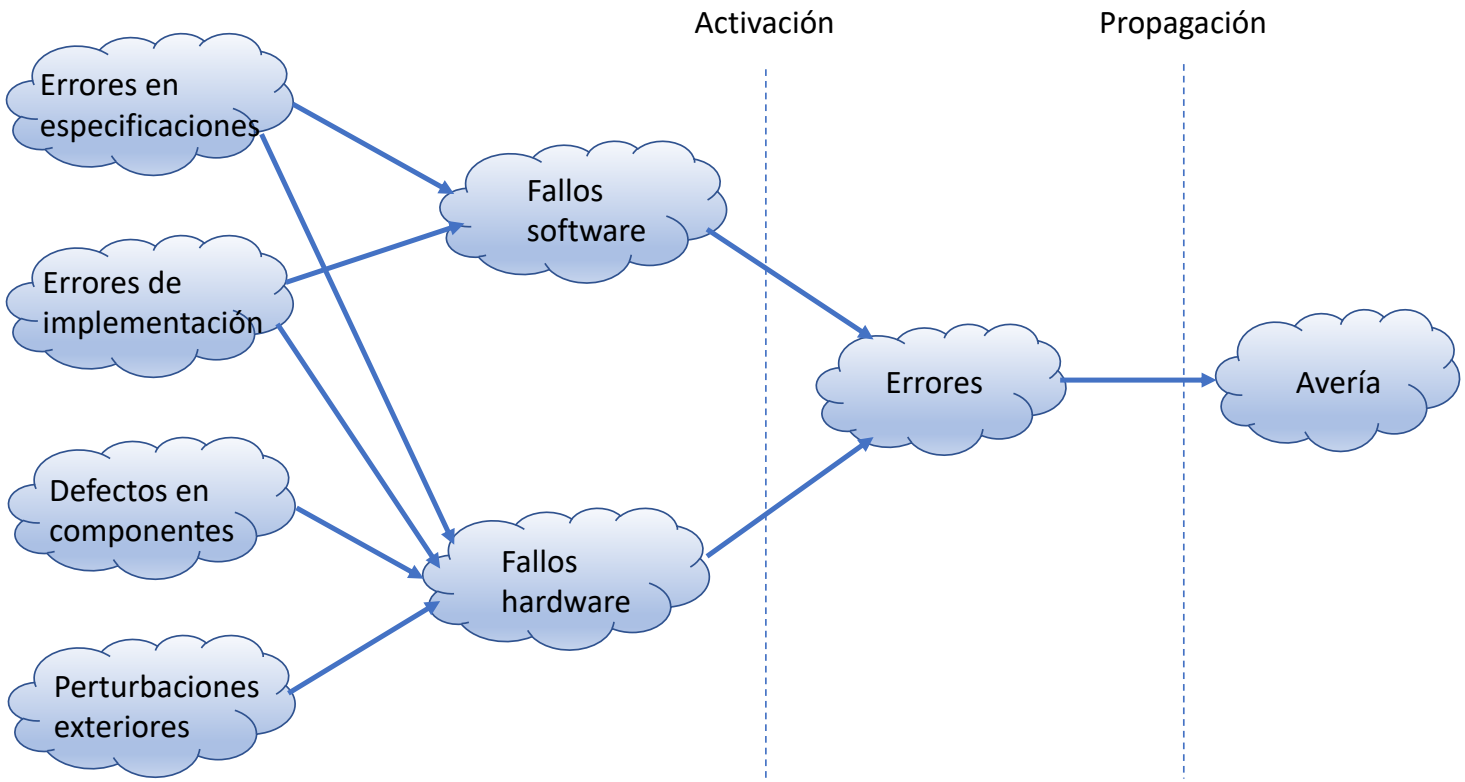
Contexto

Era exaescala

- Demandas computacionales continúan creciendo
  - Nuevos problemas
  - Big Data
- Próximo reto: computadores exaescala (10^{18} FLOPS)
  - Principal limitación: consumo energético
- Gran oportunidad para aplicaciones HPC, pero también un riesgo
- Sistemas más grandes y más complejos $\Rightarrow\downarrow\downarrow$ MTTF

MTTF (Mean Time To Failure): Tiempo medio hasta fallo

Tipo de fallos



- Atendiendo duración: transitorios, intermitentes o permanentes

Fallos en sistemas actuales

Causas de fallos

- ↓↓ Tamaño transistores ⇒↑↑ Fallos hardware
  - Radiación cósmica
  - Variaciones en la manufactura
  - Envejecimiento más rápido
- ↑↑ Complejidad programación ⇒↑↑ Fallos software
- ↑↑ Nº nodos ⇒↓↓ MTTF
  - Nodo de computación: 1 fallo cada siglo
  - Sistema con 100k nodos: 1 fallo cada 9 horas

Fallos en sistemas actuales

Blue waters

- Di Martino, C., Kalbarczyk, Z., and Iyer, R. (2016). Measuring the resiliency of extreme-scale computing environments. In Principles of Performance and Reliability Modeling and Evaluation (pp. 609-655). Springer, Cham.
  - 25k nodos, 380k núcleos, 11.6 PFLOPS
  - Aplicaciones fallidas (261 días): 9% total horas consumidas
  - Coste en electricidad: 1/2 millón dólares

- Futuros sistemas exaescala: tasa de fallos mayores
- Aplicaciones largas necesitan tolerancia a fallos
- **Tolerancia a fallos:** capacidad de continuar funcionando después de un fallo

Objetivo

MPI: estándar de facto aplicaciones HPC en sistemas distribuidos

Objetivo

- Aplicaciones MPI tolerantes a fallos
- Consideraremos:
  - Fallos hardware o software que interrumpen la ejecución de la aplicación

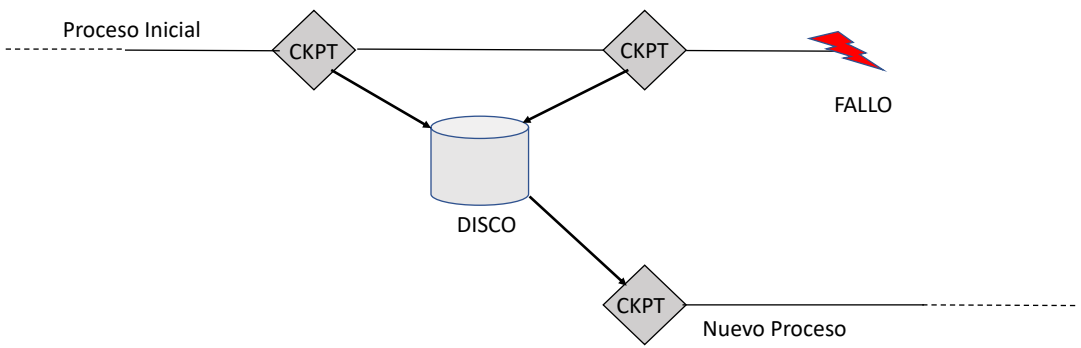
Checkpointing

Stop-and-restart

- MPI carece de soporte para tolerancia a fallos
- Solución tradicional: Stop-and-restart basado en checkpointing

Checkpointing

- Guarda de forma periódica el estado de la aplicación
- En caso de fallo, la aplicación continúa la ejecución desde el último estado guardado



Características del checkpointing

Características del checkpointing

- Granularidad, Transparencia, Coordinación

Granularidad

- A nivel de sistema:
  - Se almacena el estado completo
  - Transparente, sobrecarga alta, no portable
- A nivel de aplicación:
  - Se pueden almacenar solo datos necesarios
  - Mayor rendimiento, se necesita analizar el código

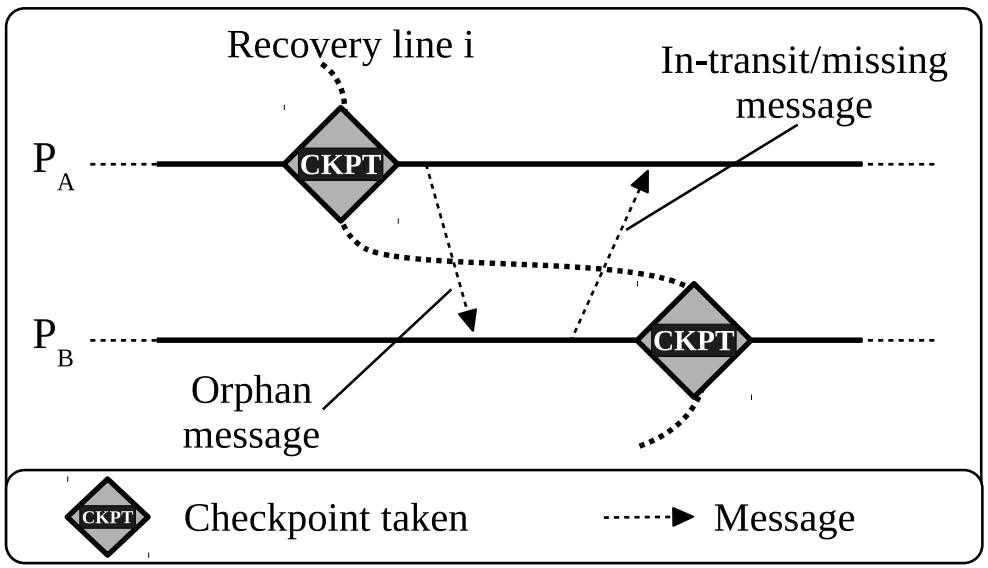
Transparencia

- Forma en la que se percibe la técnica a nivel de usuario

Características del checkpointing

Coordinación

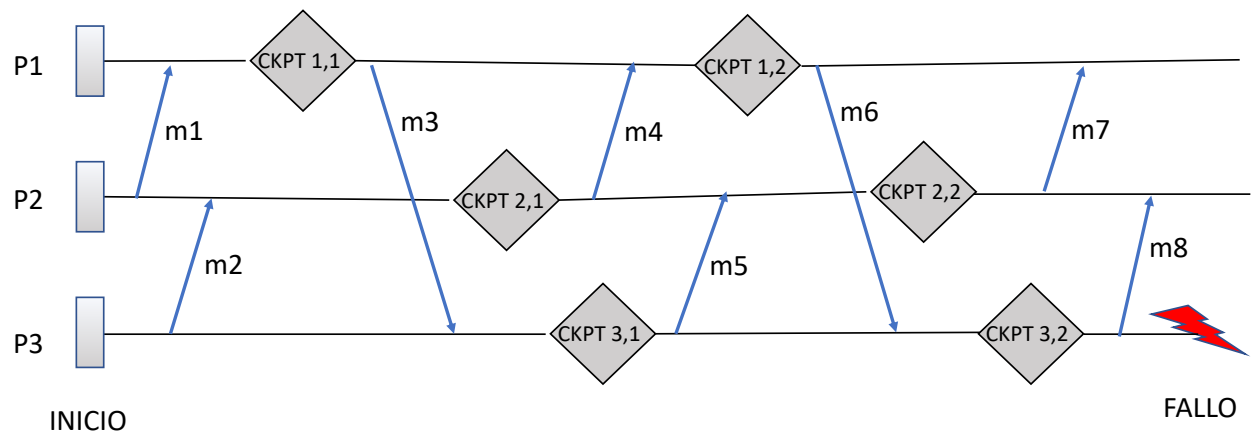
- Las aplicaciones paralelas necesitan coordinarse para evitar inconsistencias debido a las comunicaciones
  - Mensajes en tránsito: enviados pero que no han sido recibidos
  - Mensajes inconsistentes: recibidos pero que no han sido enviados



Características del checkpointing

Protocolos de coordinación

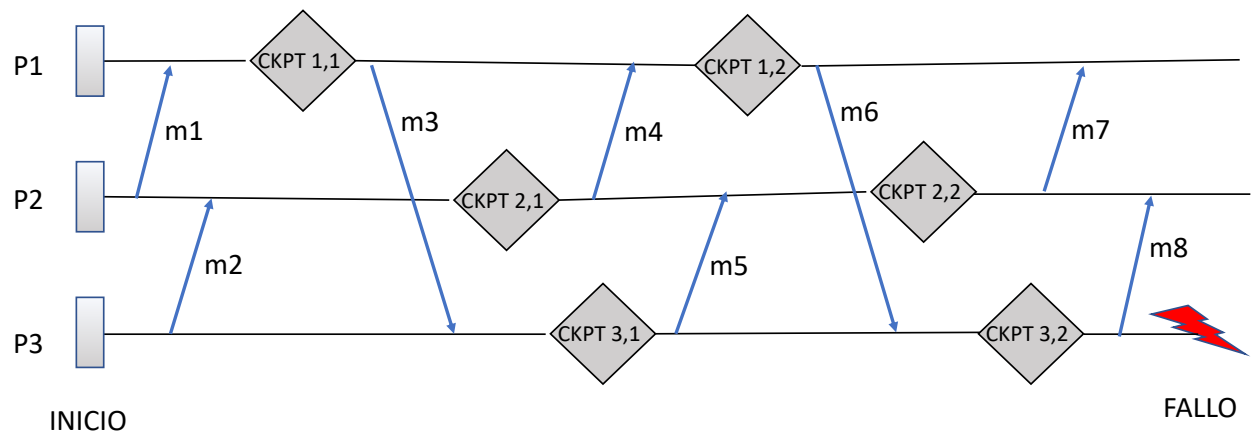
- No-coordinados
  - Baja sobrecarga en una ejecución sin fallos
  - Susceptibles de *efecto dominó* en reinicio
  - Se pueden combinar con logging de mensajes
- Coordinados
  - Los procesos se sincronizan para volcar a disco
  - Mayor sobrecarga en ejecución sin fallos



Características del checkpointing

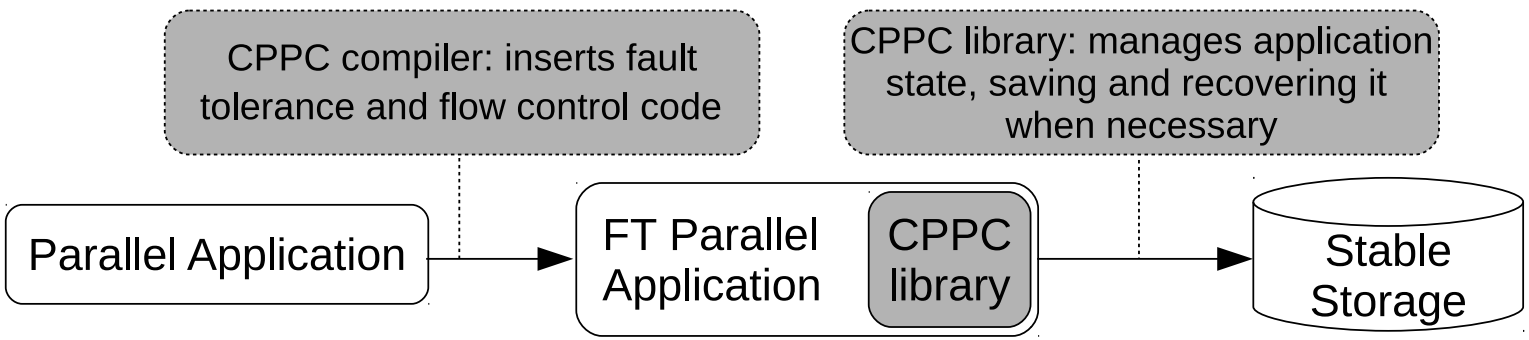
Protocolos de coordinación

- No-coordinados
  - Baja sobrecarga en una ejecución sin fallos
  - Susceptibles de *efecto dominó* en reinicio
  - Se pueden combinar con logging de mensajes
- Coordinados
  - Los procesos se sincronizan para volcar a disco
  - Mayor sobrecarga en ejecución sin fallos



CPPC — Librería + Compilador

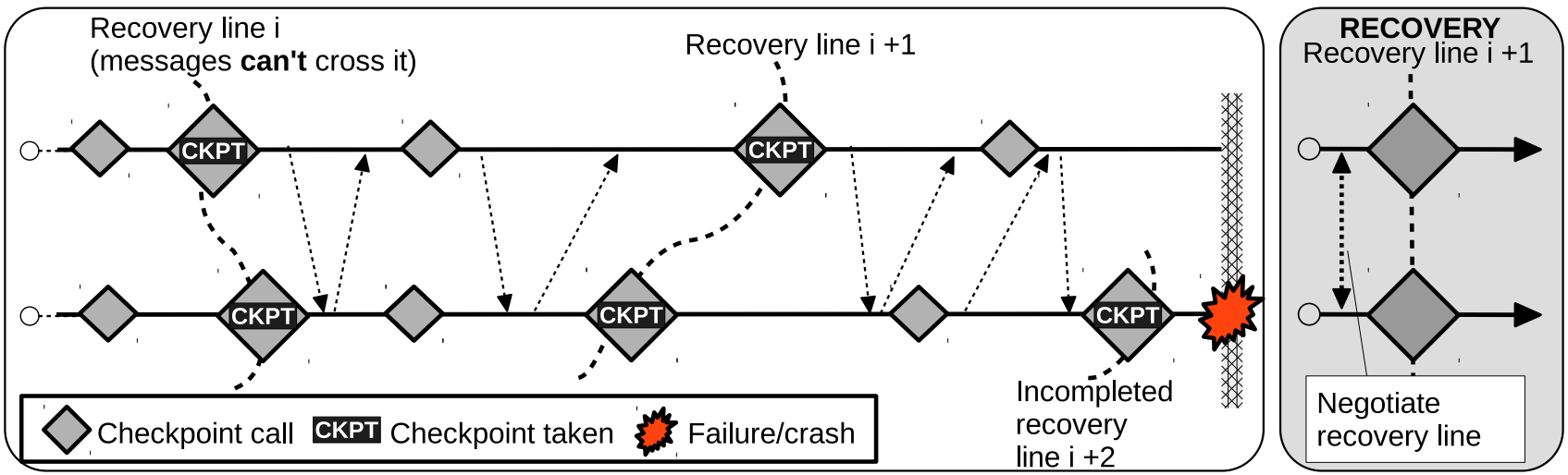
- El compilador instrumenta el código con llamadas a la librería
  - Registra las variables relevantes
  - Inserta llamadas de checkpoint
    - En puntos seguros para asegurar la consistencia
    - En los lazos más costoso (cada N llamadas) para tener una frecuencia adecuada



Volcado multi-hilo: solapa E/S y computación

Protocolo de coordinación espacial

- Procesos hacen checkpoint de forma independiente → NO sincronizaciones en tiempo de ejecución
- Llamadas de checkpoint en puntos seguros → comunicaciones no pueden cruzar la línea de recuperación



Evaluación

Puntos fuertes CPPC

- Protocolo coordinación espacial
- Tamaño ficheros checkpoint

G. Rodríguez, M.J. Martín, P. González and J. Touriño. *Analysis of Performance-impacting Factors on Checkpointing Frameworks: The CPPC Case Study.* Computer Journal, 54(11), pp.1821-1837, 2011.

Evaluación

Cluster local. 8 nodos. Cada nodo: 2×Intel Xeon con 2 núcleos y 4 GB. Red: Infiniband

	Tamaño Ckpt (MB)		Overhead FT (s)		Overhead FT+R (s)	
NPB	CPPC	BLCR	CPPC	BLCR	CPPC	BLCR
BT	623,83	753,92	0,72	7,06	20,01	33,27
CG	633,38	1205,9	0,42	26,62	21,25	50,14
EP	33,51	147,41	0,01	0,47	3,96	6,09
FT	1282,76	2033,32	19,48	60,98	30,37	87,21
IS	**2309,88**	**1774,64**	50,04	59,17	59,63	73,87
LU	264,88	390,02	0,12	29,26	8,19	44,20
MG	**500,89**	**461,57**	8,25	17,08	14,52	19.17
SP	711,74	752,85	0,17	14,33	18,68	41,53

Evaluación

Cluster local. 8 nodos. Cada nodo: 2×Intel Xeon con 2 núcleos y 4 GB. Red: Infiniband

	Tamaño Ckpt (MB)		Overhead FT (s)		Overhead FT+R (s)	
NPB	CPPC	BLCR	CPPC	BLCR	CPPC	BLCR
BT	623,83	753,92	0,72	7,06	20,01	33,27
CG	633,38	1205,9	0,42	26,62	21,25	50,14
EP	33,51	147,41	0,01	0,47	3,96	6,09
FT	1282,76	2033,32	19,48	60,98	30,37	87,21
IS	2309,88	1774,64	50,04	59,17	59,63	73,87
LU	264,88	390,02	0,12	29,26	8,19	44,20
MG	500,89	461,57	8,25	17,08	14,52	19.17
SP	711,74	752,85	0,17	14,33	18,68	41,53

Evaluación

Cluster local. 8 nodos. Cada nodo: 2×Intel Xeon con 2 núcleos y 4 GB. Red: Infiniband

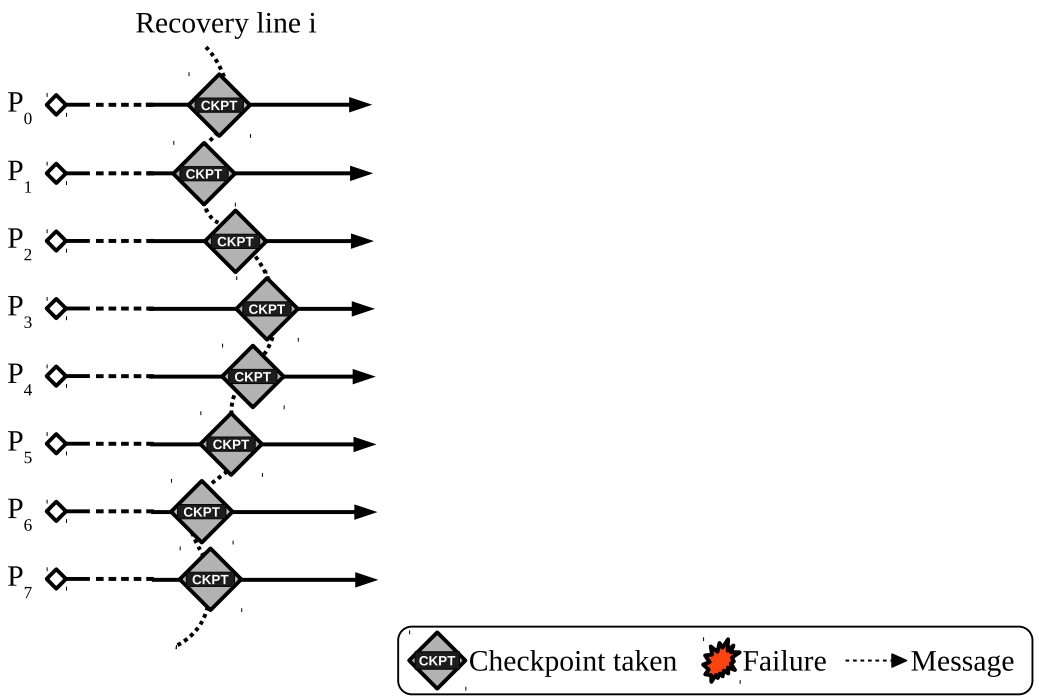
	Tamaño Ckpt (MB)		Overhead FT (s)		Overhead FT+R (s)	
NPB	CPPC	BLCR	CPPC	BLCR	CPPC	BLCR
BT	623,83	753,92	0,72	7,06	20,01	33,27
CG	633,38	1205,9	0,42	26,62	21,25	50,14
EP	33,51	147,41	0,01	0,47	3,96	6,09
FT	1282,76	2033,32	19,48	60,98	30,37	87,21
IS	2309,88	1774,64	50,04	59,17	59,63	73,87
LU	264,88	390,02	0,12	29,26	8,19	44,20
MG	500,89	461,57	8,25	17,08	14,52	19.17
SP	711,74	752,85	0,17	14,33	18,68	41,53

Evaluación

- Overhead muy ligado a tamaño fichero checkpoint
- Aplicaciones grandes -> Alta contención de E/S en sistemas de almacenamiento
- En sistemas exaescala la contención será mayor:
  - Fallos más frecuentes -> frecuencia de checkpointing mayor
  - Alto incremento en capacidad cómputo pero modesta mejora en E/S
- Estrategias para reducir tamaño:
  - Eliminación de bloques cero
  - Checkpointing incremental
  - Compresión

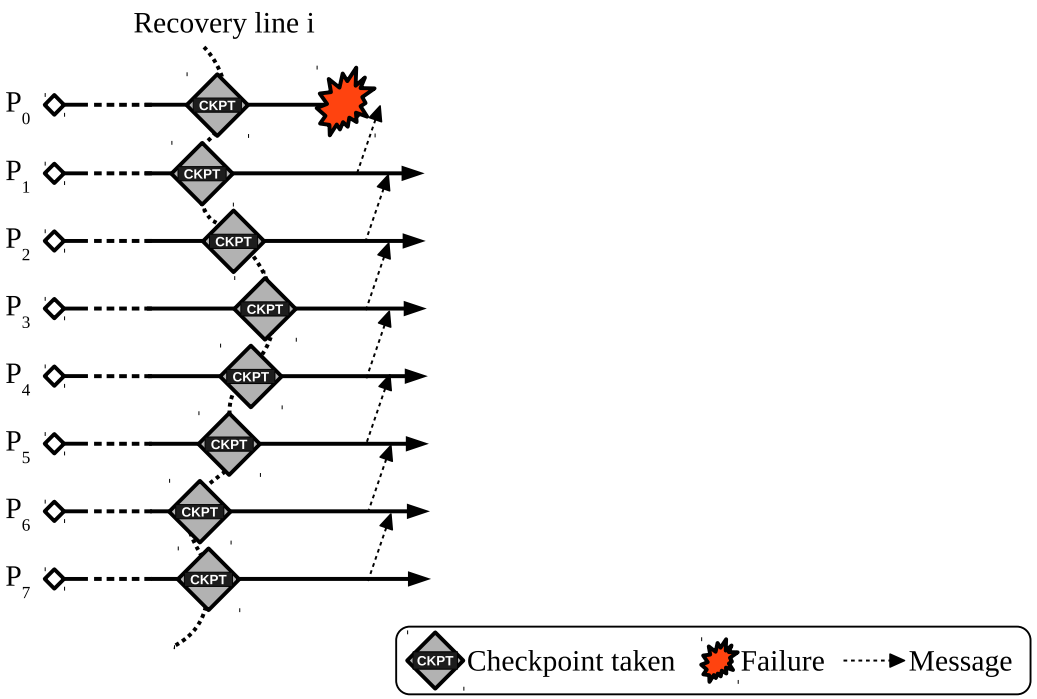
Introducción

Soluciones stop-and-restart



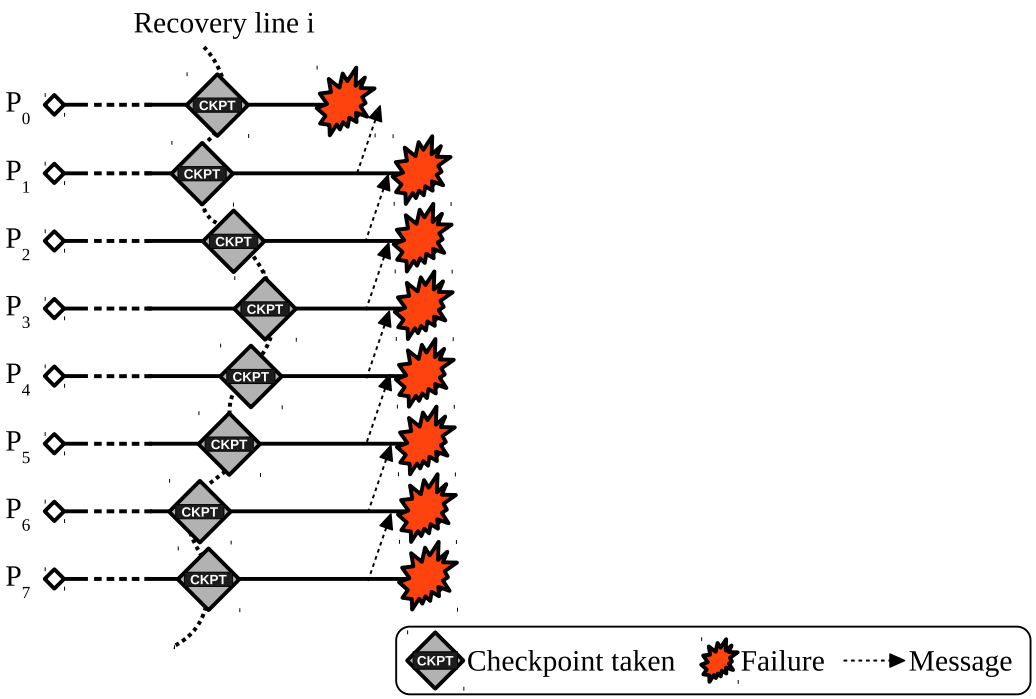
Introducción

Soluciones stop-and-restart



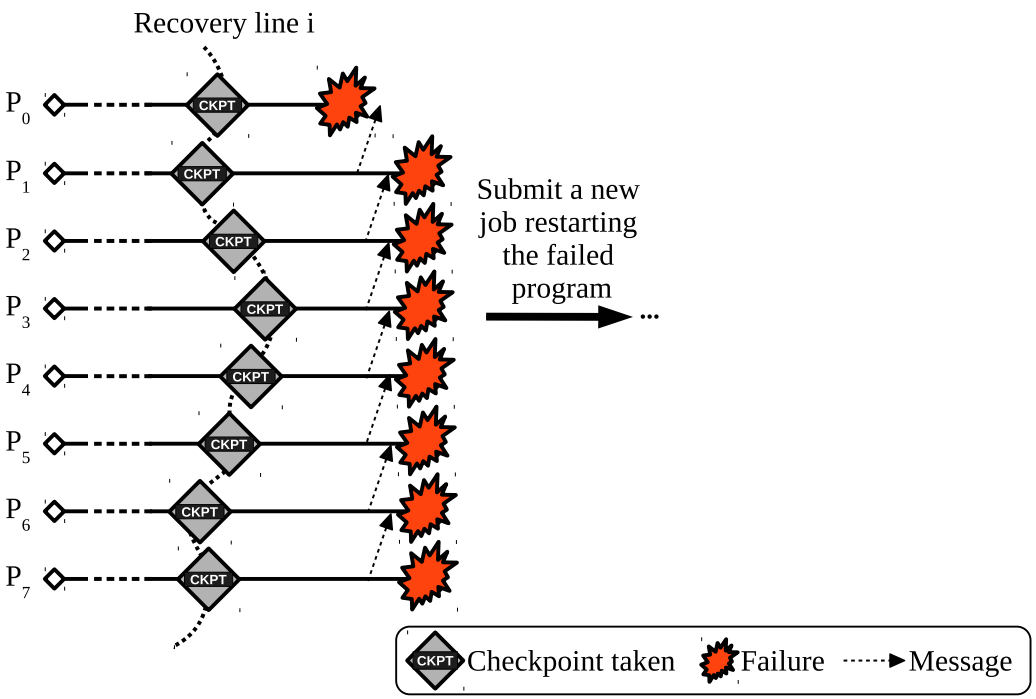
Introducción

Soluciones stop-and-restart



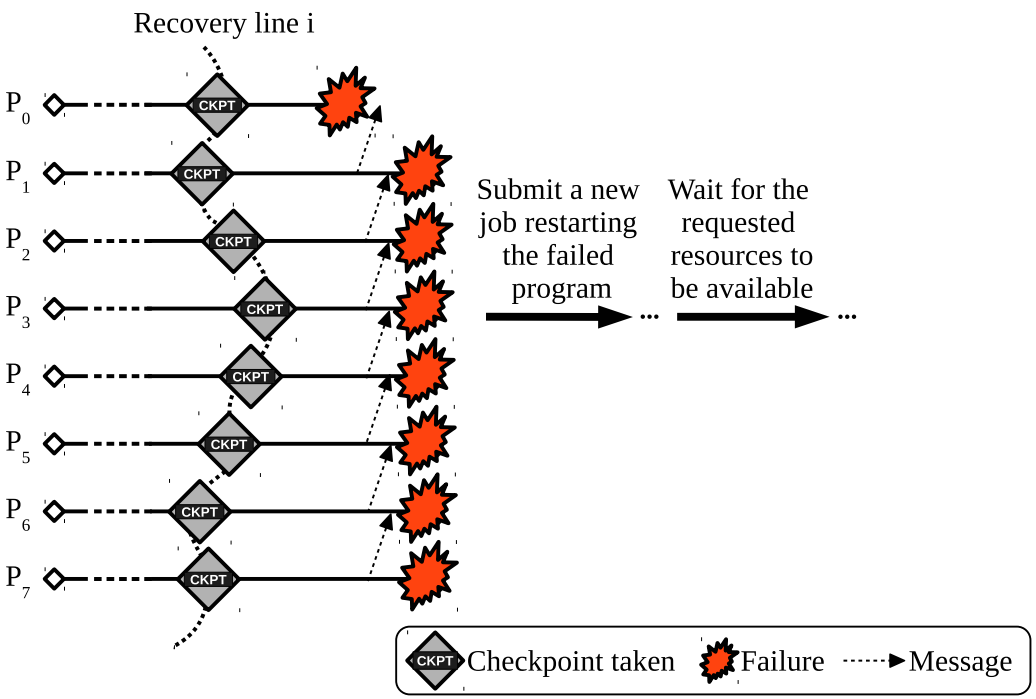
Introducción

Soluciones stop-and-restart



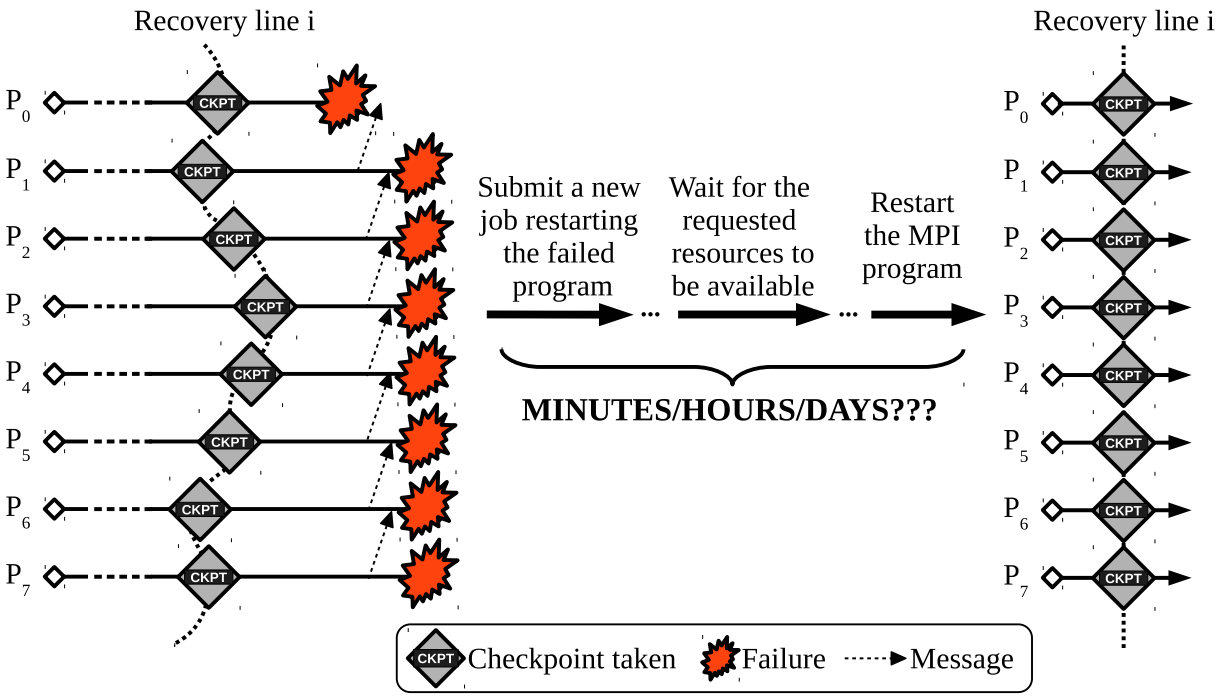
Introducción

Soluciones stop-and-restart



Introducción

Soluciones stop-and-restart



Asignación de diferentes nodos ⇒ contención de red

Introducción

- Los fallos suelen tener un impacto local
- Un restart completo es innecesario
- Para disminuir la sobrecarga ⇒ aplicaciones MPI resilientes

Aplicaciones resilientes:

Son capaces de detectar y reaccionar a fallos sin abortar su ejecución

User-level Failure Mitigation (ULFM)

User-Level Failure Mitigation (ULFM)

- Propuesto por el Fault Tolerance Working Group del MPI Forum
- Incluye nuevas semánticas para:
  - Detectar fallos
  - Propagar fallos
  - Reconfigurar comunicadores

Detección de errores

- Emplea estructuras y funciones ya presentes en estándar
  - Funciones MPI devuelven código error
  - Los manejadores de error permiten definir comportamiento en caso de error
    - MPI_ERRORS_ARE_FATAL: aborta
    - MPI_ERRORS_RETURN: devuelve código de error

User-level Failure Mitigation (ULFM)

Detección de errores

- ULFM define nuevos tipos de errores:
  - MPI_ERR_PROC_FAILED
  - MPI_ERR_PENDING
  - MPI_ERR_REVOKED
- Por defecto ULFM solo reporta errores para operaciones cuya semántica no puede ser cumplida
  - Fallos detectados por supervivientes que mantienen comunicaciones con fallidos (MPI_ERR_PROC_FAILED)
- Las operaciones MPI_ANY_SOURCE son interrumpidas con código de error especial (MPI_ERR_PENDING)

User-level Failure Mitigation (ULFM)

Propagación de fallos

- Para propagar el error al resto de los procesos se utiliza MPIX_Comm_revoke
  - Propaga el error (MPI_ERR_REVOKED)
  - Invalida el comunicador para futuras comunicaciones

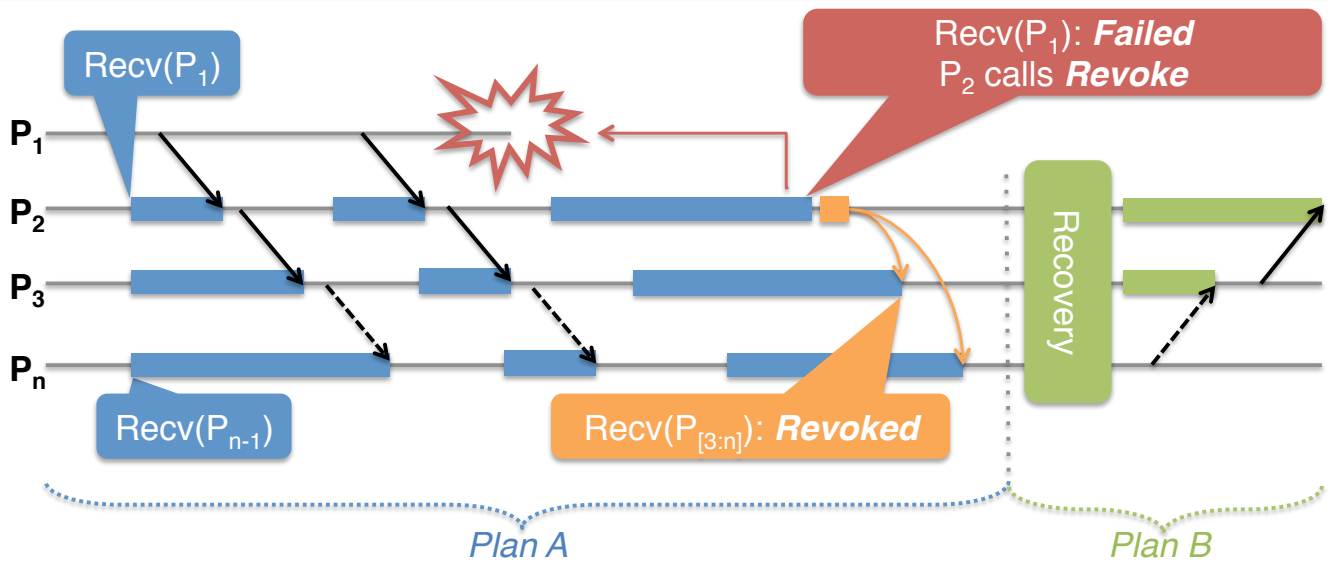


Figura cortesía de Aurelien Bouteiller (ICL, Univ. Tennessee)

User-level Failure Mitigation (ULFM)

Reconfiguración de los comunicadores

- MPIX_Comm_Shrink: Crea un nuevo comunicador sin los procesos fallidos
  - Restaura la capacidad de hacer comunicaciones colectivas
  - Aplicaciones no maleables: MPI_Comm_Spawm genera procesos de reemplazo

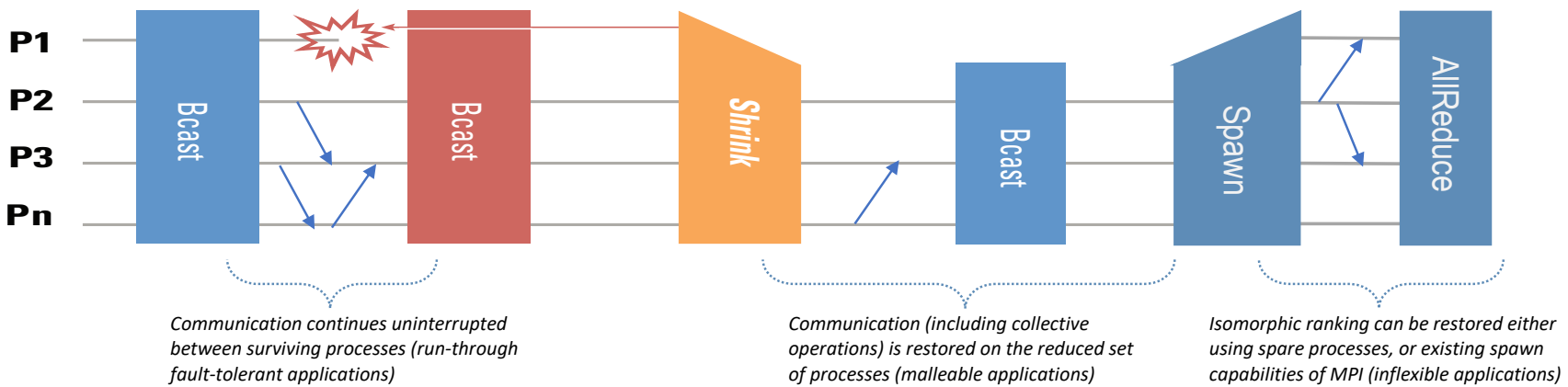


Figura cortesía de Aurelien Bouteiller (ICL, Univ. Tennessee)

Otras funciones ULFM

MPIX_Comm_agree

- Operación colectiva que permite consensuar un valor
- Permite la propagación del error de forma más estructurada
- Mete overhead en la ejecución sin fallos

MPIX_Comm_failure_ack & MPIX_Comm_failure_get_acked

- Permiten determinar qué procesos del comunicador han fallado
- Después de estas llamadas las operaciones MPI_ANY_SOURCE pueden continuar

Comparación entre Stop-and-restart y resiliencia

Detec. fallo	Stop & restart		La aplic. es abortada
	Resiliencia		Notificación fallos a todos los proc. vivos[1]
Reconstruct.	Stop & restart		La aplicación es relanzada
	Resiliencia	Shrinking	Acuerdo sobre los procesos fallidos Se eliminan proc. fallidos del comunic. Rebalanceo de la carga
		Non-shrinking	Acuerdo sobre los proc. fallidos Se eliminan proc. fallidos del comunic. Se generan nuevos procesos (spawn)[2] Se reconstruye el comunicador
Restart	Stop & restart		Recomputación desde el último ckpt
	Resiliencia	Vuelta atrás global	Recomputación desde el último ckpt
		Vuelta atrás local	Recomputación de tareas fallidas
		Sin vuelta atrás	Recomputación no necesaria

[1] En algunos casos es suficiente con notificar a un subconjunto de procesos vivos
[2] O activación de procesos de repuesto ya existentes

Comparación entre Stop-and-restart y resiliencia

Detec. fallo	Stop & restart		La aplic. es abortada
	Resiliencia		Notificación fallos a todos los proc. vivos[1]
Reconstruct.	Stop & restart		La aplicación es relanzada
	Resiliencia	Shrinking	Acuerdo sobre los procesos fallidos Se eliminan proc. fallidos del comunic. Rebalanceo de la carga
		Non-shrinking	Acuerdo sobre los proc. fallidos Se eliminan proc. fallidos del comunic. Se generan nuevos procesos (spawn)[2] Se reconstruye el comunicador
Restart	Stop & restart		Recomputación desde el último ckpt
	Resiliencia	Vuelta atrás global	Recomputación desde el último ckpt
		Vuelta atrás local	Recomputación de tareas fallidas
		Sin vuelta atrás	Recomputación no necesaria

[1] En algunos casos es suficiente con notificar a un subconjunto de procesos vivos
[2] O activación de procesos de repuesto ya existentes

Comparación entre Stop-and-restart y resiliencia

Evaluación

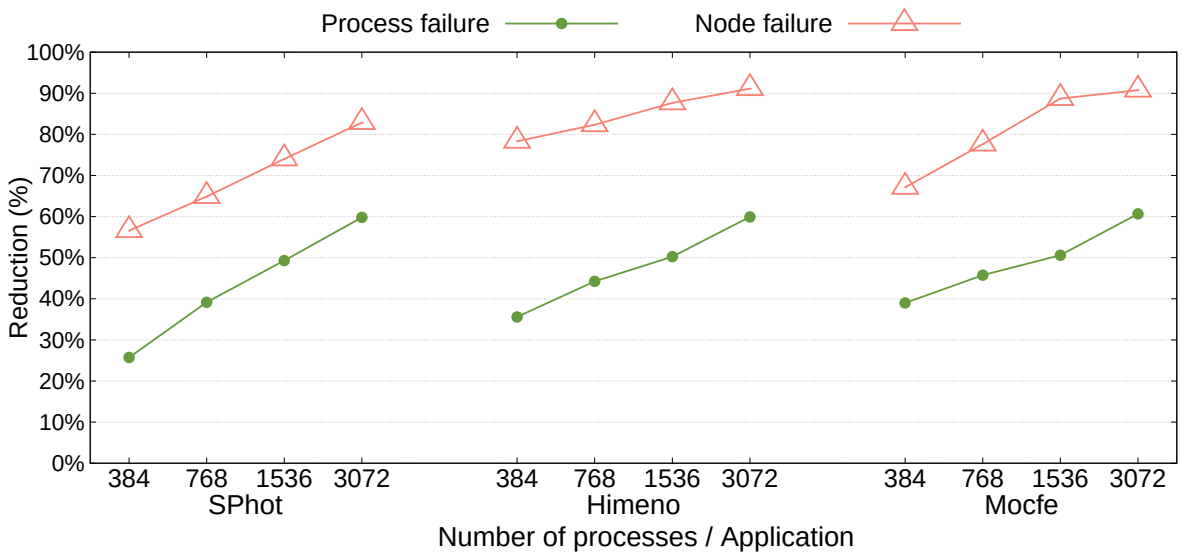
- Tiempos detección Stop-and-restart vs resiliencia

Sistema & Aplicaciones

- Supercomputador FinisTerrae-II del CESGA
  - Nodos: 2 $\times$ Intel Xeon E5-2680 $\rightarrow$ 24 núcleos & 128GB RAM
  - Red: InfiniBand FDR 56Gb/s
- 3 aplicaciones:
  - Himeno: resolutor ecuación Poisson
  - Mocfe: simula procedimientos principales de código de método de características 3D
  - Sphot: transporte de fotones 2D (benchmarks ASC Sequoia)
- 1 proceso MPI por núcleo

Comparación entre Stop-and-restart y resiliencia

N. Losada, M.J. Martín and P. González . *Assessing resilient versus stop-and-restart fault-tolerant solutions in MPI applications.*The Journal of Supercomputing 73, 1 (2017), 316–329.



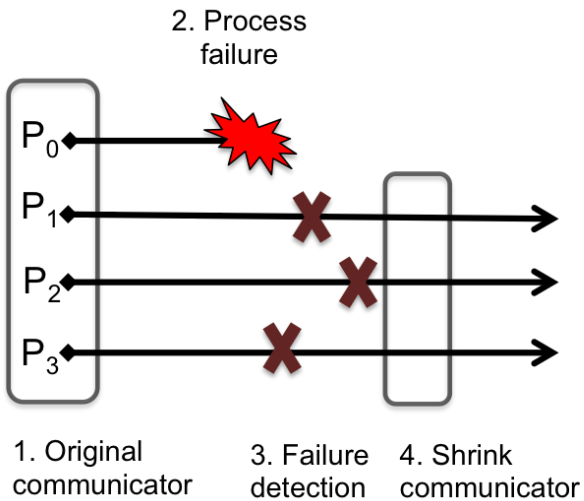
Reducción media cuando falla un proceso: 47%
Reducción media cuando falla un nodo: 79%

Comparación entre Stop-and-restart y resiliencia

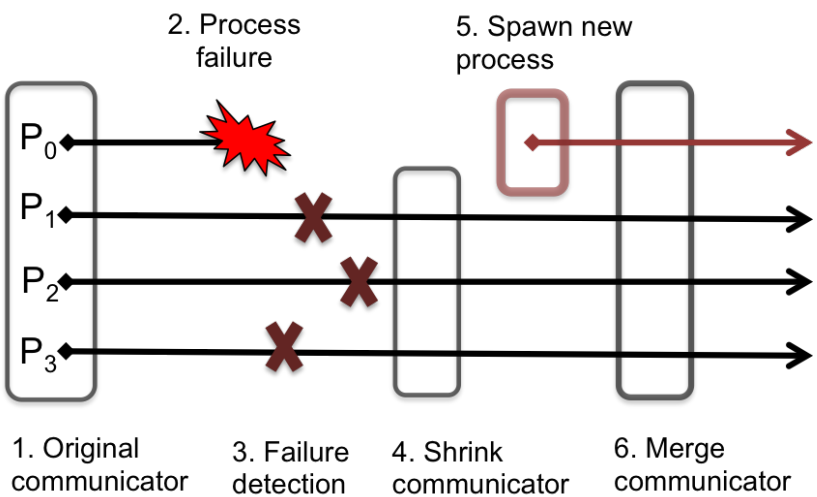
Detec. fallo	Stop & restart		La aplic. es abortada
	Resiliencia		Notificación fallos a todos los proc. vivos[1]
Reconstruct.	Stop & restart		La aplicación es relanzada
	Resiliencia	Shrinking	Acuerdo sobre los procesos fallidos Se eliminan proc. fallidos del comunic. Rebalanceo de la carga
		Non-shrinking	Acuerdo sobre los proc. fallidos Se eliminan proc. fallidos del comunic. Se generan nuevos procesos (spawn)[2] Se reconstruye el comunicador
Restart	Stop & restart		Recomputación desde el último ckpt
	Resiliencia	Vuelta atrás global	Recomputación desde el último ckpt
		Vuelta atrás local	Recomputación de tareas fallidas
		Sin vuelta atrás	Recomputación no necesaria

[1] En algunos casos es suficiente con notificar a un subconjunto de procesos vivos
[2] O activación de procesos de repuesto ya existentes

Comparación entre Stop-and-restart y resiliencia



Shrinking

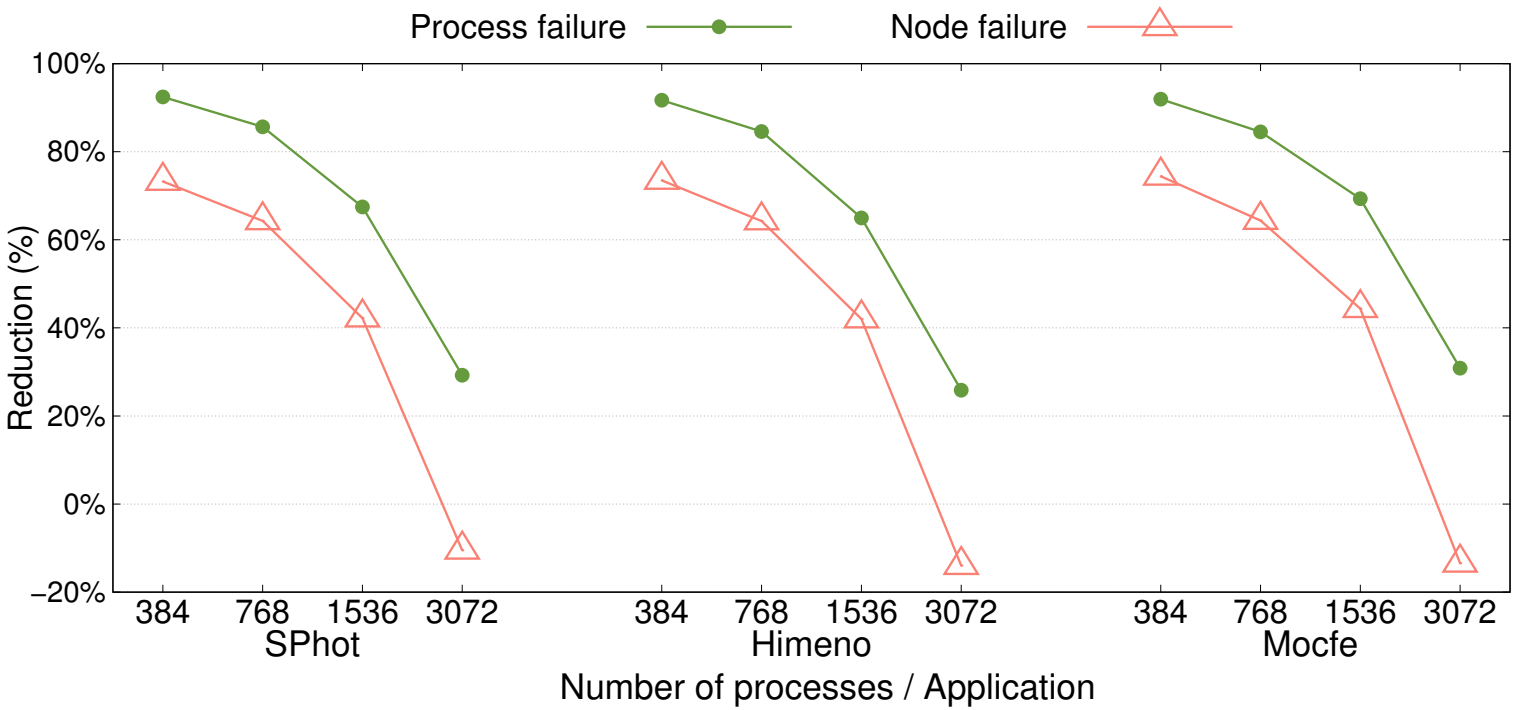


Non-shrinking

Comparación entre Stop-and-restart y resiliencia

Reducción tiempo operaciones reconstrucción con ULFM

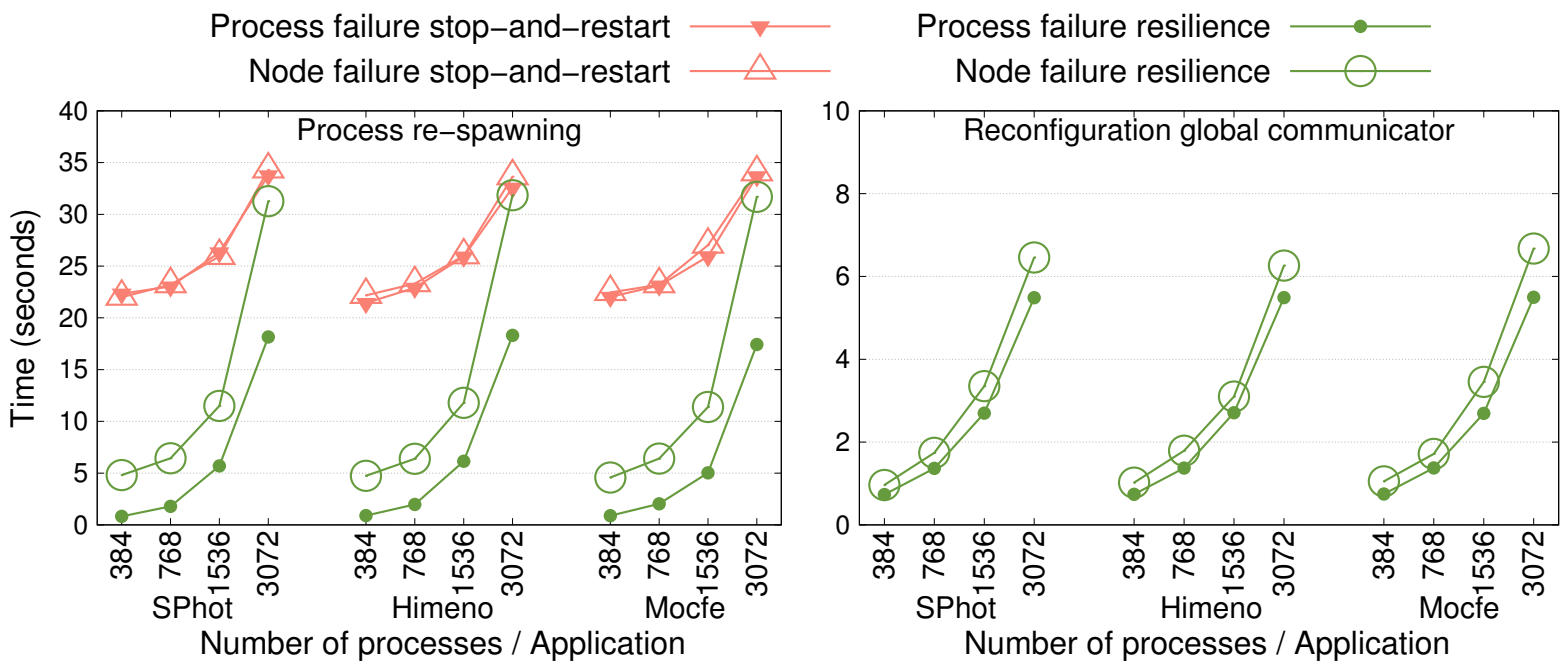
Aplicación MPI resiliente non-shrinking vs stop-and-restart



Comparación entre Stop-and-restart y resiliencia

Tiempo operaciones más costosas de la reconstrucción

Spawn de nuevos procesos y reconstrucción comunicadores



Comparación entre Stop-and-restart y resiliencia

Detec. fallo	Stop & restart		La aplic. es abortada
	Resiliencia		Notificación fallos a todos los proc. vivos[1]
Reconstruct.	Stop & restart		La aplicación es relanzada
	Resiliencia	Shrinking	Acuerdo sobre los procesos fallidos Se eliminan proc. fallidos del comunic. Rebalanceo de la carga
		Non-shrinking	Acuerdo sobre los proc. fallidos Se eliminan proc. fallidos del comunic. Se generan nuevos procesos (spawn)[2] Se reconstruye el comunicador
Restart	Stop & restart		Recomputación desde el último ckpt
	Resiliencia	Vuelta atrás global	Recomputación desde el último ckpt
		Vuelta atrás local	Recomputación de tareas fallidas
		Sin vuelta atrás	Recomputación no necesaria

[1] En algunos casos es suficiente con notificar a un subconjunto de procesos vivos
[2] O activación de procesos de repuesto ya existentes

Aplicaciones resilientes que reducen el número de procesos

Detec. fallo	Stop & restart		La aplic. es abortada
	Resiliencia		Notificación fallos a todos los proc. vivos[1]
Reconstruct.	Stop & restart		La aplicación es relanzada
	Resiliencia	Shrinking	Acuerdo sobre los procesos fallidos Se eliminan proc. fallidos del comunic. Rebalanceo de la carga
		Non-shrinking	Acuerdo sobre los proc. fallidos Se eliminan proc. fallidos del comunic. Se generan nuevos procesos (spawn)[2] Se reconstruye el comunicador
Restart	Stop & restart		Recomputación desde el último ckpt
	Resiliencia	Vuelta atrás global	Recomputación desde el último ckpt
		Vuelta atrás local	Recomputación de tareas fallidas
		Sin vuelta atrás	Recomputación no necesaria

[1] En algunos casos es suficiente con notificar a un subconjunto de procesos vivos
[2] O activación de procesos de repuesto ya existentes

Ejemplos

S. Pauli, P. Arbenz, and C. Schwab. *Intrinsic fault tolerance of multilevel monte carlo methods.* Journal of Parallel and DistributedComputing 84(2015), 24–36

- Los métodos MC se basan en el muestreo aleatorio repetido
- Versión tolerante a fallos:
  - Recuperación sin vuelta atrás y utilizando un menor número de procesos (shrinking)
  - Usan ULFM para detectar los fallos y continuar el cálculo con los procesos vivos
  - Muestras afectadas por fallo se descartan
  - Muy bajo overhead a costa de degradación resultados

Ejemplos

I. Laguna, D.F. Richards, T. Gamblin, M. Schulz and B.R. de Supinski. *Evaluating user-level fault tolerance for MPI applications.* In European MPI Users' Group Meeting (2014), ACM, p. 57

- Aplicación de dinámica molecular
- Recuperación con vuelta atrás global utilizando un menor número de procesos
- En caso de fallo:
  - Se usa ULFM para detectar fallos, revocar y hacer el *shrink* de los comunicadores
  - El estado de la aplicación se recupera desde el último checkpoint
    - Utiliza checkpointing almacenado en memoria
  - La carga es rebalanceada entre los procesos vivos

Ejemplos

Aplicaciones maestro-esclavo

- Se utiliza ULFM para detectar esclavos fallidos
- Solo las tareas fallidas necesitan ser recalculadas
- Se puede usar un menor número de procesos y la carga se rebalancea dinámicamente
- Recuperación local hacia atrás usando un menor número de procesos

Aplicaciones resilientes que NO reducen el número de procesos

Detec. fallo	Stop & restart		La aplic. es abortada
	Resiliencia		Notificación fallos a todos los proc. vivos[1]
Reconstruct.	Stop & restart		La aplicación es relanzada
	Resiliencia	Shrinking	Acuerdo sobre los procesos fallidos Se eliminan proc. fallidos del comunic. Rebalanceo de la carga
		Non-shrinking	Acuerdo sobre los proc. fallidos Se eliminan proc. fallidos del comunic. Se generan nuevos procesos (spawn)[2] Se reconstruye el comunicador
Restart	Stop & restart		Recomputación desde el último ckpt
	Resiliencia	Vuelta atrás global	Recomputación desde el último ckpt
		Vuelta atrás local	Recomputación de tareas fallidas
		Sin vuelta atrás	Recomputación no necesaria

[1] En algunos casos es suficiente con notificar a un subconjunto de procesos vivos
[2] O activación de procesos de repuesto ya existentes

Aplicaciones resilientes que NO reducen el número de procesos

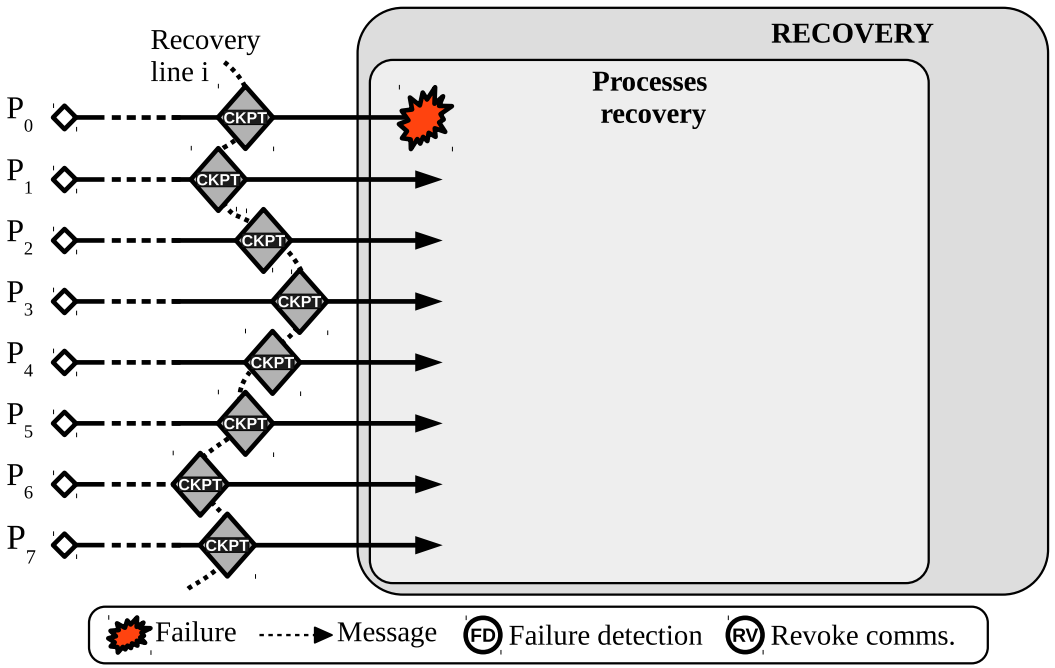
Detec. fallo	Stop & restart		La aplic. es abortada
	Resiliencia		Notificación fallos a todos los proc. vivos[1]
Reconstruct.	Stop & restart		La aplicación es relanzada
	Resiliencia	Shrinking	Acuerdo sobre los procesos fallidos Se eliminan proc. fallidos del comunic. Rebalanceo de la carga
		Non-shrinking	Acuerdo sobre los proc. fallidos Se eliminan proc. fallidos del comunic. Se generan nuevos procesos (spawn)[2] Se reconstruye el comunicador
Restart	Stop & restart		Recomputación desde el último ckpt
	Resiliencia	Vuelta atrás global	Recomputación desde el último ckpt
		Vuelta atrás local	Recomputación de tareas fallidas
		Sin vuelta atrás	Recomputación no necesaria

[1] En algunos casos es suficiente con notificar a un subconjunto de procesos vivos
[2] O activación de procesos de repuesto ya existentes

Nuestra propuesta

- Vuelta atrás local para aplicaciones MPI SPMD genéricas
  - Sólo los procesos afectados por el fallo van atrás
- Combina ULFM, CPPC y el logging de mensajes
- Colaboración con Innovative Computing Laboratory (Univ. Tennessee)

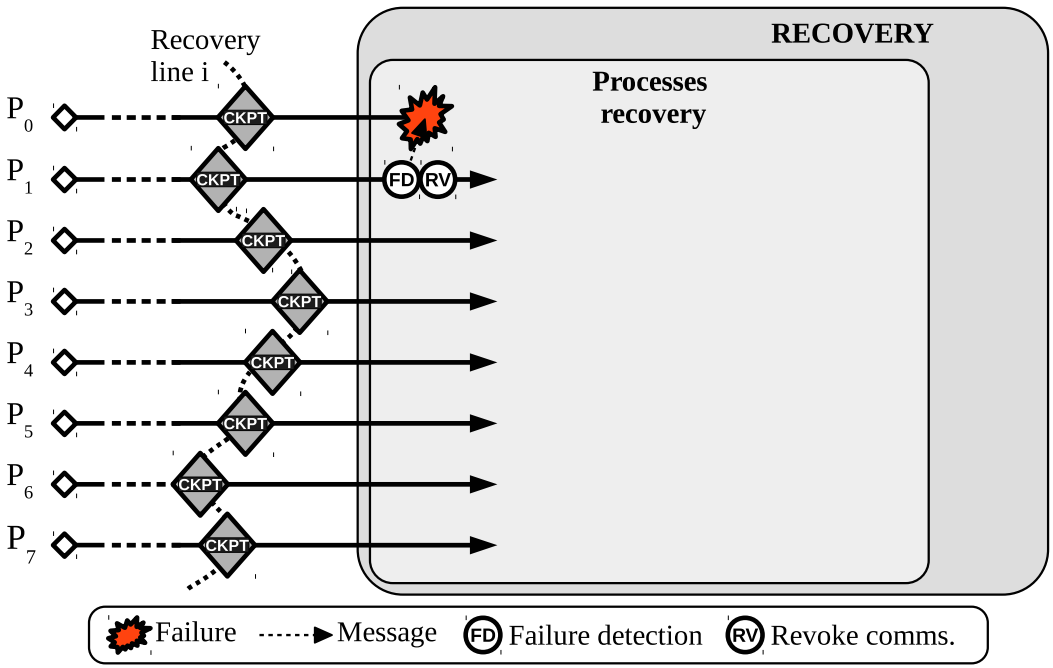
Protocolo de vuelta atrás local



Primer paso: recuperación de los procesos

- Se utilizan las funcionalidades de ULFM

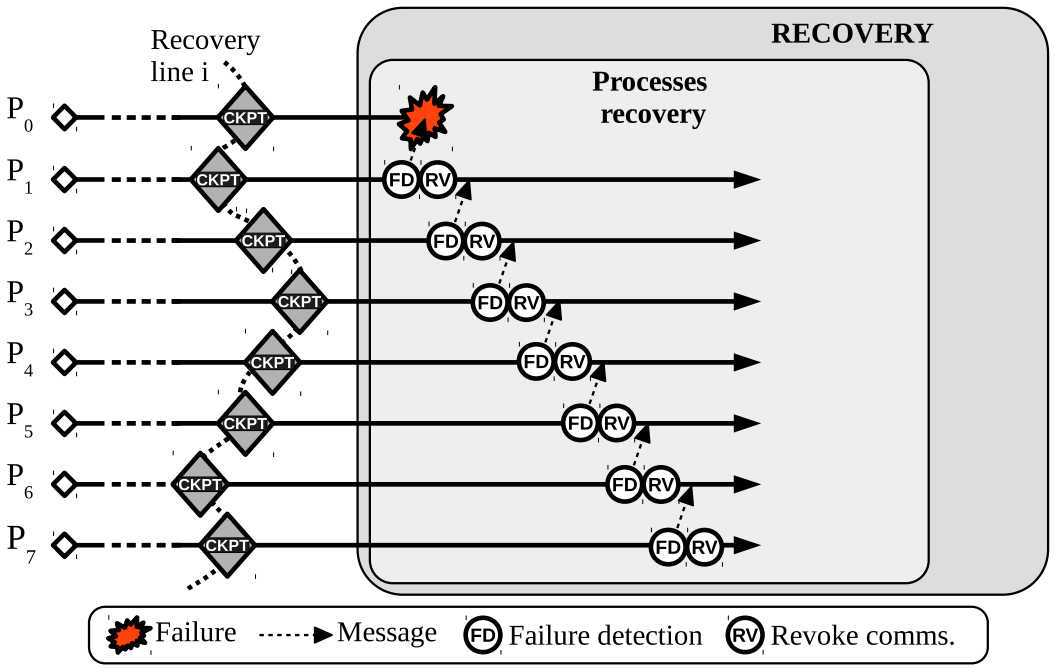
Protocolo de vuelta atrás local



Primer paso: recuperación de los procesos

- Los supervivientes que se comunican con un fallido detectan el fallo y revocan comunicadores

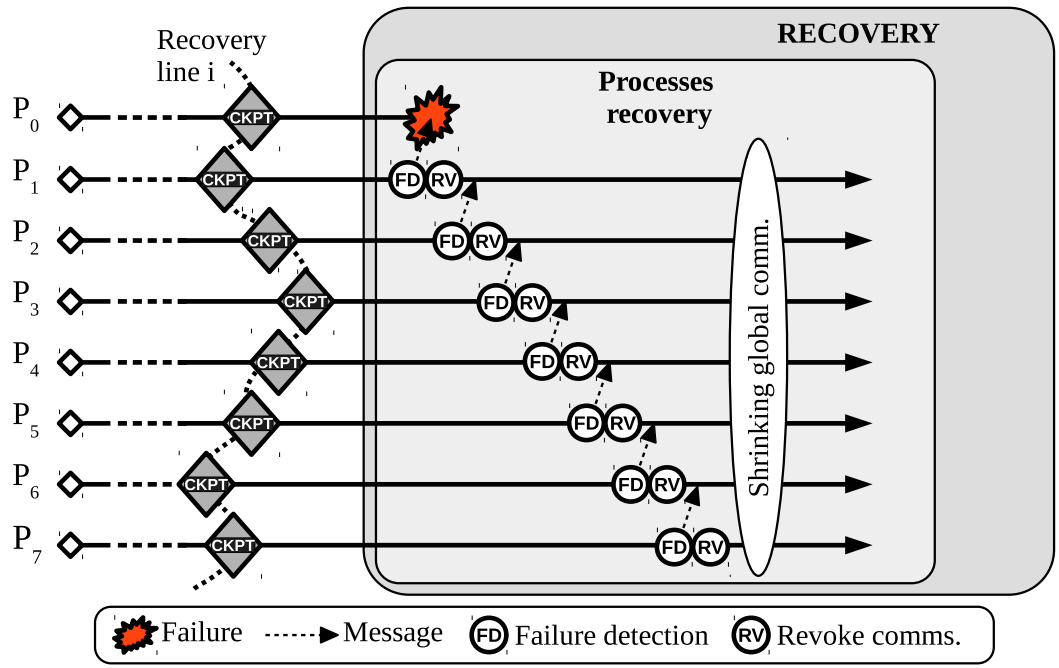
Protocolo de vuelta atrás local



Primer paso: recuperación de los procesos

- El error se propaga a todos los supervivientes

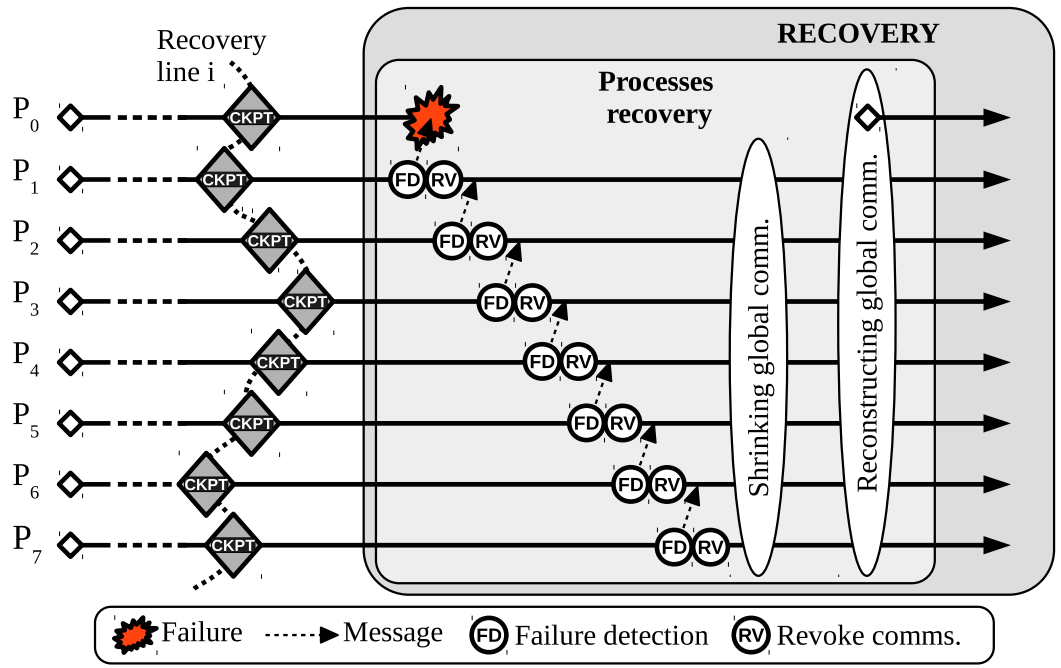
Protocolo de vuelta atrás local



Primer paso: recuperación de los procesos

- Supervivientes acuerdan procesos fallidos
- Excluyen procesos fallidos del comunicador

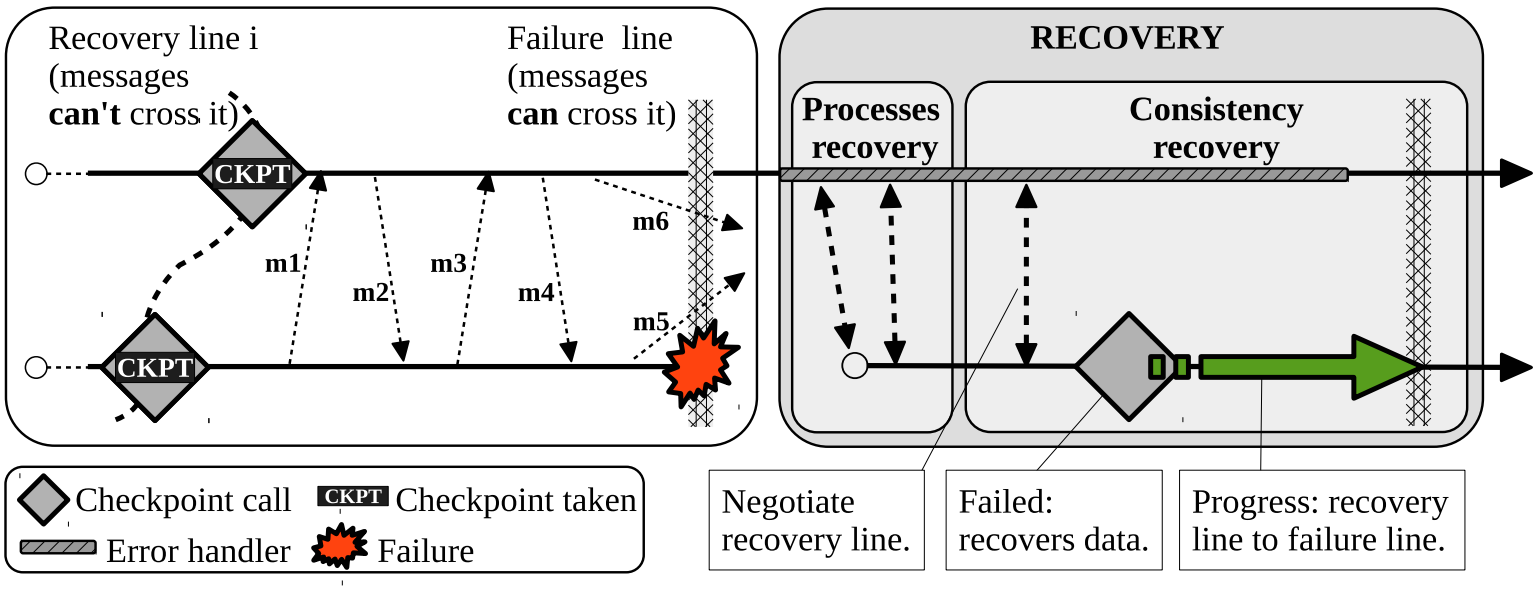
Protocolo de vuelta atrás local



Primer paso: recuperación de los procesos

- Se lanzan nuevos procesos para sustituir fallidos
- Se reconstruye el comunicador

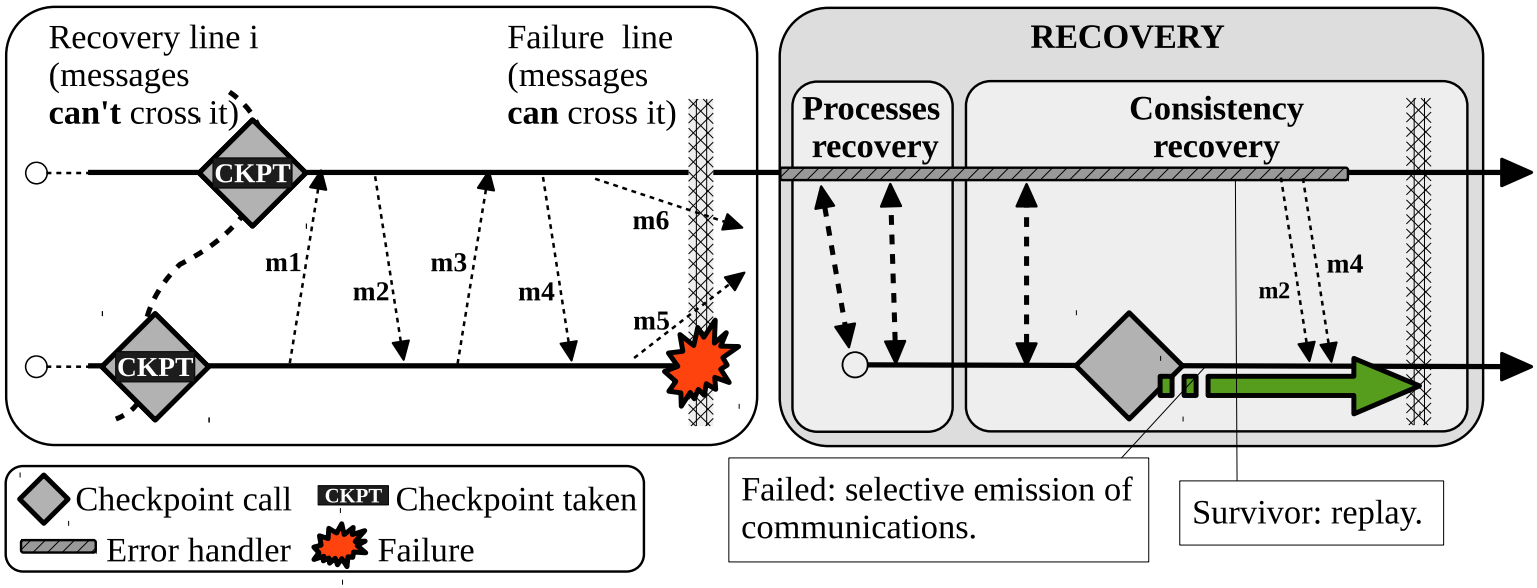
Protocolo de vuelta atrás local



Segundo paso: Recuperación de la consistencia

- Se negocia la linea de recuperación
- Procesos fallidos:
  - Recuperan los datos del fichero de checkpointing
  - Avanzan desde la línea de recuperación a la línea de fallos

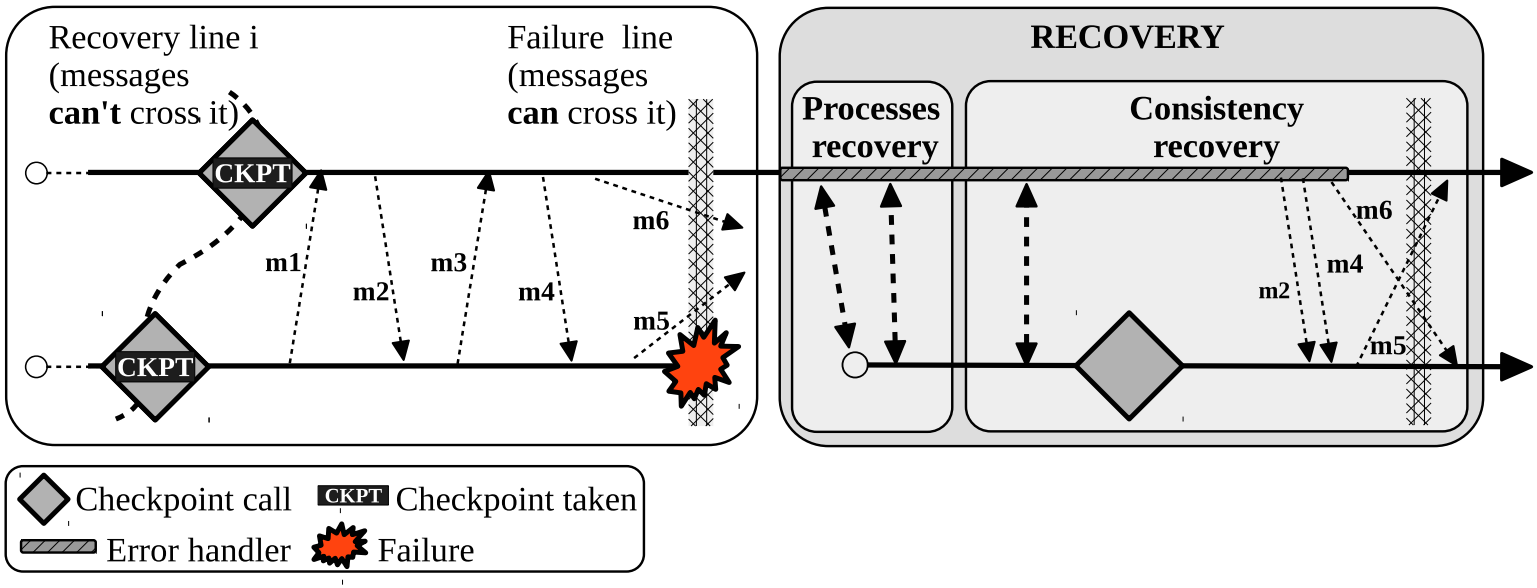
Protocolo de vuelta atrás local



Segundo paso - Progreso procesos fallidos

- Para llegar al mismo estado:
  - Algunos mensajes tienen que ser enviados de nuevo
  - Otros tienen que ser saltados

Protocolo de vuelta atrás local



Segundo paso - Consistencia supervivientes

- Hay que reenvíar también las comunicaciones interrumpidas por los fallos

Implementación

Logging de los mensajes

- Logging a nivel de sistema para comunicaciones punto a punto (VProtocol)
- Logging a nivel de aplicación para comunicaciones colectivas

Seguimiento de los mensajes

- Utilizamos los SSIDs (Sender Sequence IDs) de los mensajes

Implicaciones del protocolo de coordinación espacial de CPPC

- Comunicaciones no pueden cruzar línea recuperación $\Rightarrow$ Logs pueden ser borrados en líneas recuperación ($\downarrow$ uso memoria).

Implementación

Logging de los mensajes

- Logging a nivel de sistema para comunicaciones punto a punto (VProtocol)
- Logging a nivel de aplicación para comunicaciones colectivas

Seguimiento de los mensajes

- Utilizamos los SSIDs (Sender Sequence IDs) de los mensajes

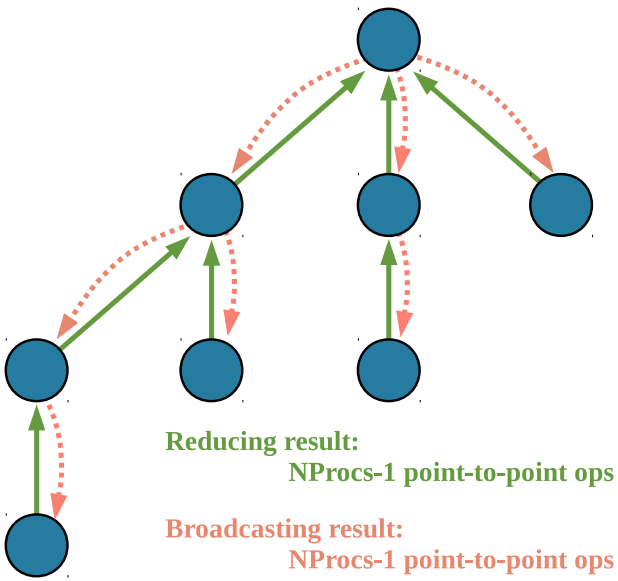
Implicaciones del protocolo de coordinación espacial de CPPC

- Comunicaciones no pueden cruzar línea recuperación $\Rightarrow$ Logs pueden ser borrados en líneas recuperación ($\downarrow$ uso memoria).

Implementación

Comunicaciones colectivas - Log a nivel de aplicación

- No hace log de comunicaciones punto-a-punto internas
- Reduce tamaño log en colectivas con copias intermedias



Ejemplo: MPI_Allreduce con árbol binomial

- Log comunicaciones punto-a-punto:
  $2 \times (Nprocs - 1) \times Buffsize$ bytes
- Log a nivel de aplicación:
  $Nprocs \times Buffsize$ bytes

Evaluación experimental

Sistema & Applicaciones

- Supercomputador FinisTerrae-II del CESGA.
  - Nodos: 2 $\times$ Intel Xeon E5-2680 $\rightarrow$ 24 núcleos & 128GB RAM
  - Red: InfiniBand FDR 56Gb/s
- Checkpoints almacenados en disco (Lustre sobre InfiniBand)
- 4 aplicaciones: Himeno, Mocfe, Sphot, Tealeaf
  - Tealeaf: ecuación lineal conducción calor (proyecto Mantevo)
  - 1 proceso MPI por núcleo

Evaluación experimental

Tiempos de ejecución originales en minutos

	48P	96P	192P	384P	768P
HIMENO	18.48	9.22	4.76	2.45	1.56
MOCFE	20.49	8.26	4.75	2.41	1.48
SPHOT	16.38	8.25	3.78	2.25	1.48
TEALEAF	17.50	9.31	4.28	2.35	1.79

Pruebas

- Comparación vuelta atrás local vs. vuelta atrás global
  - Ambas implementan resiliencia usando ULFM
  - Checkpointing al 40% de la ejecución

Evaluación experimental

Tiempos de ejecución originales en minutos

	48P	96P	192P	384P	768P
HIMENO	18.48	9.22	4.76	2.45	1.56
MOCFE	20.49	8.26	4.75	2.41	1.48
SPHOT	16.38	8.25	3.78	2.25	1.48
TEALEAF	17.50	9.31	4.28	2.35	1.79

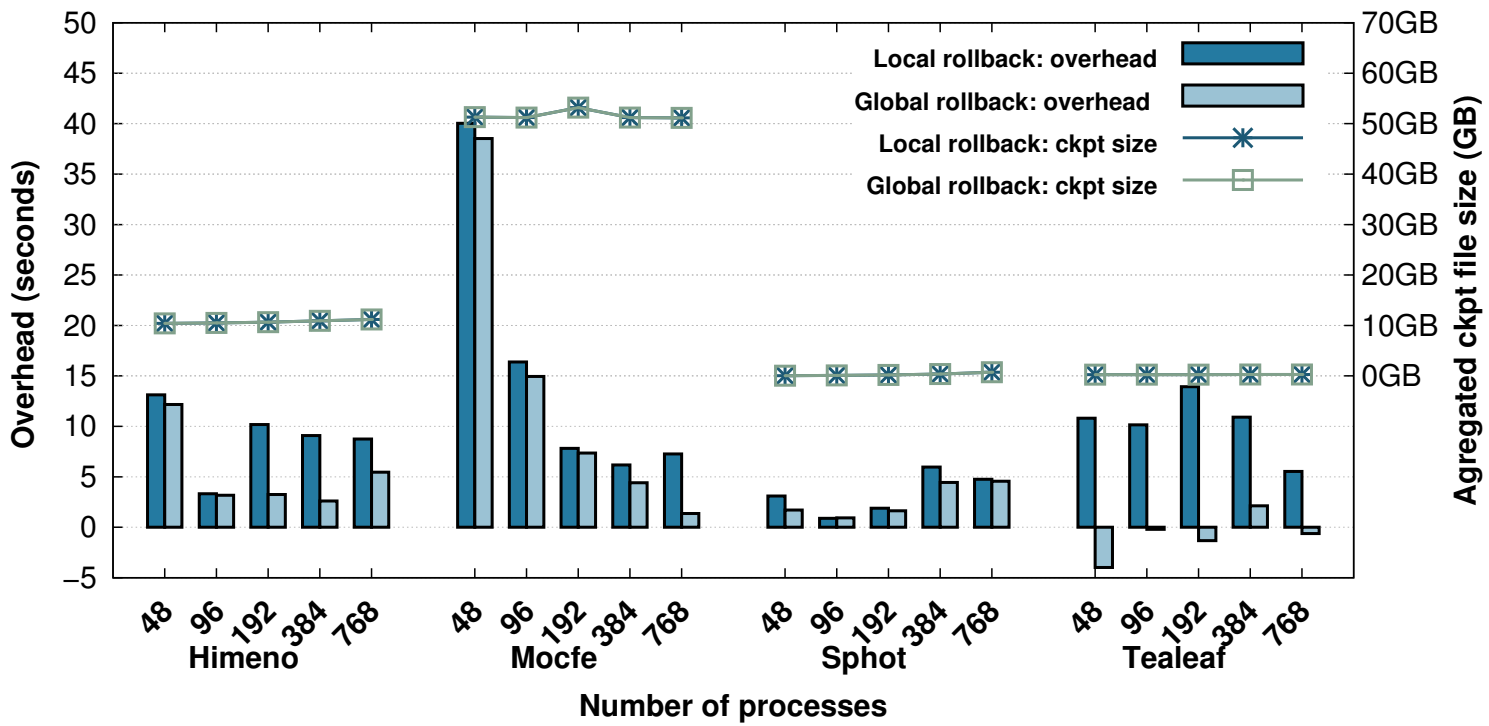
Pruebas

- Comparación vuelta atrás local vs. vuelta atrás global
  - Ambas implementan resiliencia usando ULFM
  - Checkpointing al 40% de la ejecución

Evaluación experimental

Overhead en la ejecución sin fallos

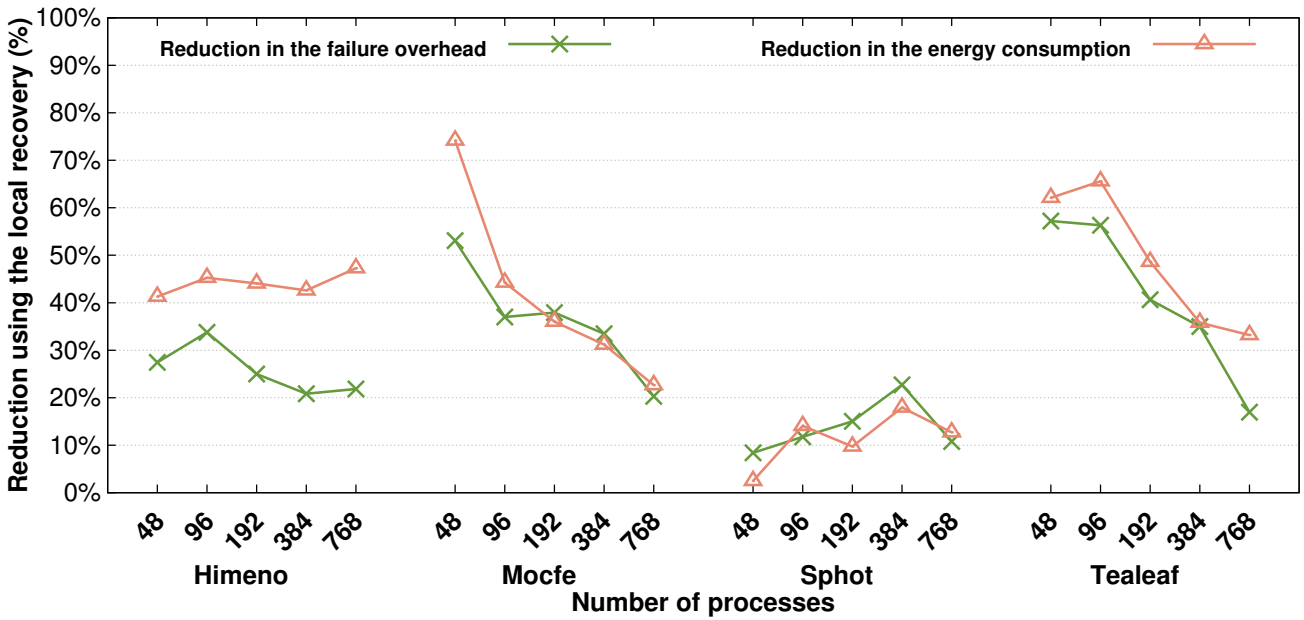
Vuelta atrás local (ckpt+log) vs Vuelta atrás global (ckpt)



Evaluación experimental

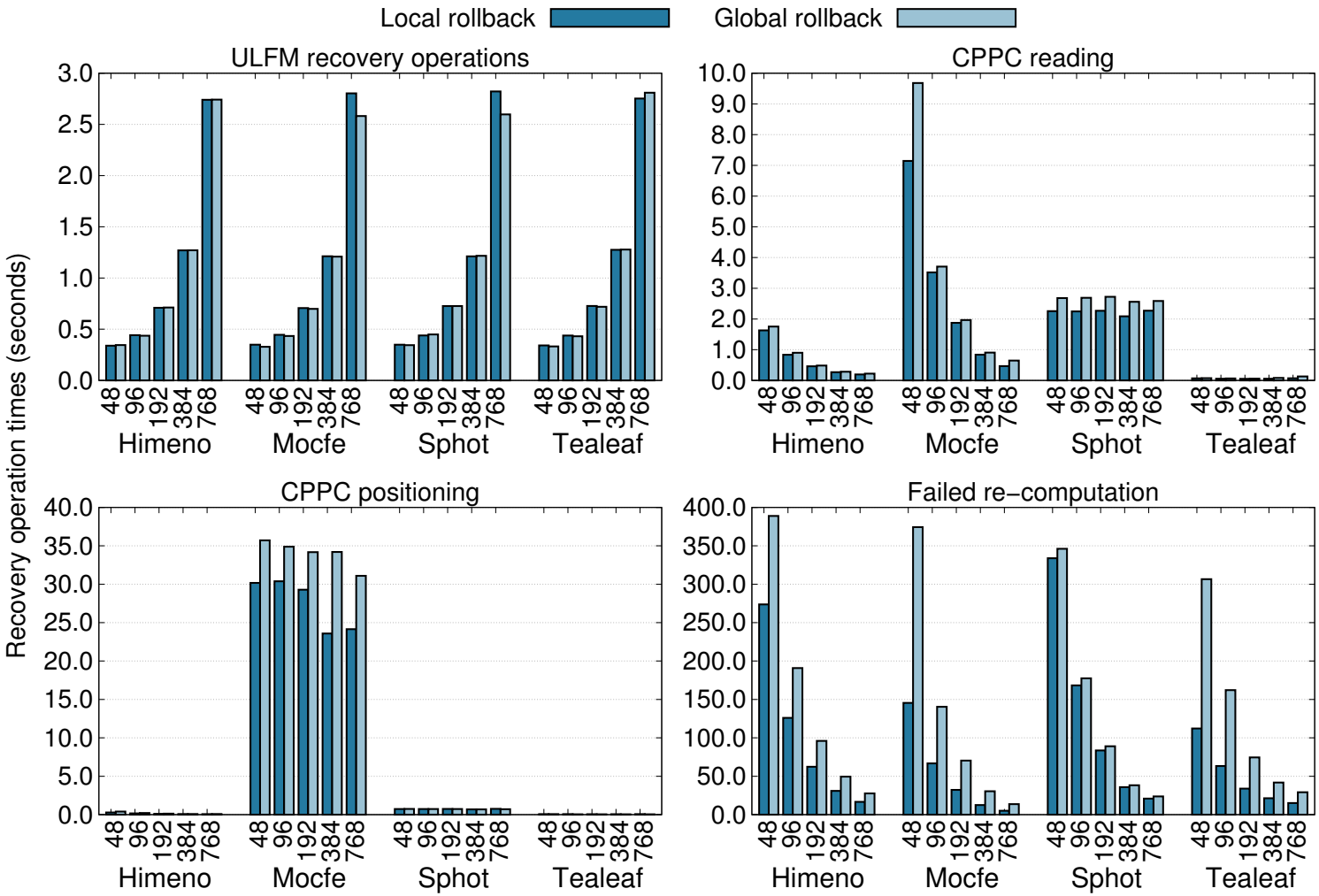
Overhead en la ejecución con fallos

Fallo al 75% de la ejecución (matando el último proceso).
Reducción (cuando se usa vuelta atrás local vs global) en:
Tiempo de ejecución extra y energía consumida



Reducción tiempo extra: 30% en media

Evaluación experimental



Conclusiones

Conclusiones

- ↑ Núm. procesadores ⇒ ↓ MTTF
- Soluciones tradicionales:
  - Stop-and-restart⇒ ↑ overhead
- Alternativa:
  - Aplicaciones resilientes
- ULFM funciones básicas de bajo nivel para construir aplicaciones resilientes ⇒ No trivial
- Se necesitan librerías más alto nivel para facilitar desarrollo aplicaciones resilientes

Aplicaciones MPI resilientes

María J. Martín

Grupo de Arquitectura de Computadores
Universidade da Coruña
mariam@udc.es